# ColrC Documentation

An easy to use C library for linux terminal colors/escape-codes.

Version 0.3.6

Generated by Doxygen

# Contents

# 0.1 Documentation

## 0.1.1 Getting Started

**ColrC** (***kuh**·lr·see*, feels like heresy) is a C library for terminal colors/escape-codes on linux.

There is also a command-line tool (`colr tool`) based on **ColrC**.

It is designed to be flexible and easy to use. Colors can be specified using defined names (`RED`, `BLUE`, etc.), 256-colors (`ext(36)`), `RGB` colors (`rgb(0, 0, 55)`), hex colors (`hex(s)`, `hex("#ff0000")`), or known names (`"aliceblue"`). These colors can be used with `fore()` and `back()` to set the foreground/background colors (`fore(RED)`, `back(WHITE)`). Styles are specified with their defined names (`style(BRIGHT)`).

Strings can be joined, replaced, colorized, and justified using a few functions/macros. `fore()`, `back()`, and `style()` are mostly optional and position doesn't matter.

Ownership in **ColrC** is easy to remember. Strings (`char*`) are yours, everything else belongs to **ColrC**. If you create a **ColrC** object with one of the `Colr*` macros to use inside of the `colr*` macros (notice the casing), it will be released. The resulting strings that are returned from the `colr*` macros will not be released. You must `free()` those.

If you use `colr_print` or `colr_puts` you won't have to manage the resulting string either.

### 0.1.1.1 Including

You must include colr.h and compile colr.c along with your program.

```
#include "colr.h"

int main(void) {
    // Simple usage:
    char* s = colr("Hello from ColrC!", fore("blueviolet"), back(WHITE));
    if (!s) return EXIT_FAILURE;
    puts(s);
    // Or just:
    colr_puts(Colr("Hello again!", fore(rgb(255, 0, 0)),
      back("#ffff00")));

    // Fancier functions:
    char* s2 = colr_replace(
        s,
        "Hello",
        Colr_join(
            " ",
            Colr_cat(
                Colr("Good", fore(rgb(0, 0, 255)), back(RESET)),
                Colr("bye", fore(CYAN), style(BRIGHT))
            ),
            "and",
            Colr("good luck", style(UNDERLINE))
        )
    );
    free(s);
    if (!s2) return EXIT_FAILURE;
    puts(s2);
    free(s2);

    return EXIT_SUCCESS;
}
```

There are plenty of examples in the `documentation`, and `on this page`.

#### 0.1.1.2 Compiling

ColrC uses a couple glibc features, which may not be compatible with your system. Most linux distros are compatible.

The colr.h header defines `_GNU_SOURCE` if it's not already defined (see `man feature_test_macros`).

*Be sure to include **libm** (the math library) when compiling*:

```
gcc -std=c11 -c myprogram.c colr.c -o myexecutable -lm
```

#### 0.1.1.3 Files

The only two files that are needed to use ColrC are colr.h and colr.c.

| Name | Description |
|---|---|
| colr.h | The interface to ColrC. |
| colr.c | Where ColrC is implemented. This must be compiled/linked with your program. |

You can also create a shared library (`libcolr.so`) for your system. Clone the repo and run the make target:

```
make lib
```

If you link the library (and `libm`), you will only need to include the header (`colr.h`):

```
gcc -std=c11 -c myprogram.c -o myexecutable -lm -lcolr
```

#### 0.1.1.4 Example Usage

You use colr_cat(), colr_join(), and Colr(), along with fore(), back(), and style() to build colorized strings. There are some print-related functions, for quick building/printing of colorized strings (colr_puts() and colr_print()).

```
#include "colr.h"

int main(int argc, char** argv) {
    // Print-related macros, using Colr() to build colorized text:
    puts("\nColrC supports ");
    colr_puts(Colr_join(
        ", ",
        Colr("basic", fore(WHITE)),
        Colr("extended (256)", fore(ext(155))),
        Colr("rgb", fore(rgb(155, 25, 195))),
        Colr("hex", fore(hex("#ff00bb"))),
```

```
        Colr("extended hex", fore(ext_hex("#ff00bb"))),
        Colr("color names", fore("dodgerblue"), back("aliceblue")),
        Colr("and styles.", style(BRIGHT))
    ));

    colr_puts(
        "Strings and ",
        Colr("colors", fore(LIGHTBLUE)),
        " can be mixed in any order."
    );

    // Create a string, using colr(), instead of colr_puts() or colr_print().
    char* mystr = colr("Don't want to print this.", style(UNDERLINE));
    printf("\nNow I do: %s\n", mystr);
    free(mystr);

    // Concatenate existing strings with ColrC objects.
    // Remember that the colr macro free ColrC objects, not strings.
    // So I'm going to use the Colr* macros inside of this call (not colr*).
    char* catted = colr_cat(
        "Exhibit: ",
        Colr("b", fore(BLUE)),
        "\nThe ColorText/Colr was released."
    );
    puts(catted);
    free(catted);

    // Create a ColorText, on the heap, for use with colr_cat(), colr_print(),
    // or colr_puts().
    ColorText* ctext = NULL;
    if (argc == 1) {
        ctext = Colr("<nothing>", fore(RED));
    } else {
        ctext = Colr(argv[1], fore(GREEN));
    }
    char* userstr = colr_cat("Argument: ", ctext);
    puts(userstr);
    // colr_cat() already called ColorText_free(ctext).
    free(userstr);

    // Create a joined string (a "[warning]" label).
    char* warning_label = colr_join(Colr("warning", fore(YELLOW)), "[", "]");
    // Simulate multiple uses of the string.
    for (int i = 1; i < 4; i++) printf("%s This is #%d\n", warning_label, i);
    // Okay, now we're done with the colorized string.
    free(warning_label);

    // Colorize an existing string by replacing a word.
    char* logtext = "[warning] This is an awesome warning.";
    char* colorized = colr_replace(
        logtext,
        "warning",
        Colr("warning", fore(YELLOW))
    );
    // Failed to allocate for new string?
    if (!colorized) return EXIT_FAILURE;
    puts(colorized);
    // You have to free the resulting string.
    free(colorized);

    // Or colorize an existing string by replacing a regex pattern.
    colorized = colr_replace_re(
        logtext,
```

```
        "\\[\\w+\\]",
        Colr_join(
            Colr("ok", style(BRIGHT)),
            "(",
            ")"
        ),
        REG_EXTENDED
    );
    if (!colorized) return EXIT_FAILURE;
    puts(colorized);
    free(colorized);

    // Or maybe you want to replace ALL of the occurrences?
    char* logtext2 = "[warning] This is an awesome warning.";
    // There is also a colr_replace_re_all() if you'd rather use a regex pattern.
    char* colorizedall = colr_replace_all(
        logtext2,
        "warning",
        Colr("WARNING", fore(YELLOW))
    );
    // Failed to allocate for new string?
    if (!colorizedall) return EXIT_FAILURE;
    puts(colorizedall);
    // You have to free the resulting string.
    free(colorizedall);

}
```

#### 0.1.1.4.1 Example Files

For all examples, check the documentation. Here is a table of the most common usage examples:

| Name | Example |
|---|---|
| Colr | Colr_example.c |
| colr_cat | colr_cat_example.c |
| colr_join | colr_join_example.c |
| colr_replace | colr_replace_example.c |
| colr_replace_re | colr_replace_re_↩ example.c |
| fore | fore_example.c |
| back | back_example.c |
| style | style_example.c |

All of the examples can be built with the `examples` target:

```
make examples
```

You can then run the executables in `./examples` manually, with the make target (`make runexamples`), or with the example runner:

```
./examples/run_example.sh [NAME_PATTERN...]
```

There is also a "snippet runner" that can build and run arbitrary C code snippets, mainly used for building and running all example code snippets found in the ColrC source code itself:

```
./tools/snippet.py --examples
```

To see a list of source-based examples in the terminal you can run:

```
./tools/snippet.py --listnames [NAME_PATTERN]
```

To view the source code for those examples, you can run:

```
./tools/snippet.py --listexamples [NAME_PATTERN]
```

#### 0.1.1.5 Why

ColrC is the C version of `Colr` (a python library) and it's less-flexible cousin `Colr.sh`. The programming styles vary because C doesn't allow easy method chaining, and instead leans towards nested function calls.

This is an attempt to create a flexible and easy version for C.

#### 0.1.1.6 Future

In the future there may be a shared library or a python extension based on ColrC, but for now I'm finishing out the basic features and testing.

## 0.2 Development

### 0.2.1 ColrC Development

If you are looking to send a pull request, or compile the `colrc` tool yourself, there are a few things you might need to know. These subpages contain information about compiling, testing, system dependencies, and anything else relevant to working on **ColrC** itself.

They are not required reading for an average user of colr.h and colr.c.

- Dependencies: Dependencies for working on ColrC.
- Testing: How ColrC is tested.
- Make: Make targets to build/test ColrC.
- Tools: Tools to help with ColrC development.
- Examples: Examples provided by the ColrC documentation.
- Compatibility: Notes about ColrC system compatibility.

### 0.2.2 Dependencies

#### 0.2.2.1 System

To compile the `colrc` tool, or use the helper tools, you will need a few system dependencies:

- `gcc` or `clang`
  - You can use `gcc` or `clang` to compile ColrC.
  - `gcc 7.4.0+` or `clang 3.9.0+` is recommended.
- `make`
  - The main build steps are implemented in make files.
  - GNU Make 4.1+ is recommended (other versions may work).
- `libc`
  - The ColrC tests use GNU extensions, and certain ColrC features are enabled when compiled with `libc`.
  - ColrC uses `libm` to implement it's "rainbow"-related functions.
  - `libc6-dev 2.27+` is recommended.
- `python3`
  - Several scripts in `./tools` use Python.
  - Python 3.6+ is recommended.
- `bash`
  - Several scripts in `./tools` use BASH-specific features.
  - BASH 4.4+ is recommended.
- `valgrind`
  - Used for it's `memcheck` tool, to test for memory leaks in ColrC code, examples, and snippets.
- `cppcheck`
  - Used for extra linting of the ColrC source code.
- `lcov`
  - Used to generate test coverage reports.
- `doxygen`
  - Documentation for ColrC is generated with Doxygen.
  - Doxygen 1.8+ is recommended.
- `doxygen-latex`
  - Extras to generate the PDF manual.
- `texlive-lang-cyrillic`
  - Includes fonts for the PDF manual.
- `texlive-fonts-extra`
  - Includes fonts for the PDF manual.
- `texlive-latex-base`
  - Provides the `pdflatex` command to generate the PDF manual.
- `texlive-binaries`
  - Provides the `makeindex` command to generate the PDF manual.

#### 0.2.2.2 Python

There are several helper tools in the ColrC repo. They are responsible for running tests, generating documentation, running `valgrind`, and other conveniences. The python-based tools have their own dependencies:

- `colr`
  - Provides terminal colors and the `colr-run` tool.
  - This was also the inspiration for ColrC.
- `docopt`
  - Provides argument parsing.
- `easysettings`
  - Provides settings/configuration files.
- `fmtblock`
  - Provides text block formatting.
- `outputcatcher`
  - Provides stdout/stderr blocking/catching.
- `printdebug`
  - Provides debug information while running the tools.
- `pygments`
  - Provides syntax highlighting for code listed with the tools.

There is a `requirements.txt` in the `./tools` directory for easy installation of these packages (`pip install -r requirements.txt`).

### 0.2.3 Tests

#### 0.2.3.1 About

ColrC uses `snow` for testing. There are several test targets in the `makefile` that do different things. Some of them are for quick sanity-checking, some use compiler protections, and some use Valgrind. There is also a test runner (`run_tests.sh`) that provides an easy way to run tests through a wrapper program like `valgrind` or kdbg/gdb.

#### 0.2.3.2 Basic Test:

If you want to run them you will have to download/clone the source and build/run them:

```
# The default ‘test‘ target uses ‘-fsanitize‘ options, which can be slow:
make test
```

This will build all of the tests using the latest `colr.c` and run them.

#### 0.2.3.3 Memcheck Test:

You can also run the tests through `valgrind` with the `testmemcheck` target:

```
# Removes the ‘-fsanitize‘ options, to let ‘valgrind‘ do it’s thing:
make testmemcheck
```

#### 0.2.3.4 Quick Testing

During development, I usually use the `testfast` target for small changes, followed by a `testfull` to use the address sanitizer and other protection features.

```
make testfast

# After I’ve sorted out the "easy" failures:
make testfull

# And finally, before pushing changes, the "everything test".
# This is important because it ensures that all examples will compile cleanly
# and there are no leaks:
make testeverything
```

#### 0.2.3.5 Test Everything

The 'everything test' builds the colr tool and unit tests, both debug and release mode (some bugs only show up in release mode), and runs them through `valgrind` and `-fsanitize` (`libasan`).

The examples are built and ran through `valgrind`, including the examples found in the source code (see `snippet.py --examples`). This ensures that all example code is correct/runnable.

The coverage target is built (with the html report).

Finally, the binaries may be rebuilt if they are in a different state than when the process started (switch back to debug build for development).

If any of those things fail, the process is stopped and there is probably a bug worth fixing. Errors are always reported, but the noise from all of those steps can be silenced with `--quiet`.

Each of these steps has found one or more bugs in the code or documentation while developing ColrC. I don't mind running this before pushing my changes.

If you'd like to run every possible compile target, with tests and memcheck, including the example code and source-file examples (the 'everything test'):

```
make testeverything
```

#### 0.2.3.6 Test Tool

The `./test/run_tests.sh` script can run the `snow`-based tests, run `memcheck` on the examples, and run the `colrc` tool through `memcheck`. The "everything test" is implemented with this tool. Run `./test/run_test.sh -h` to see options for it.

### 0.2.4 Make

#### 0.2.4.1 ColrC Make Targets

ColrC is built using `make`, and though there are plenty of targets in the main directory, `./test`, and `./examples`, only a few are needed to make confident changes to ColrC. Most test targets have a `quiet` version that only shows failures in the terminal.

The typical workflow looks like this:

```
# Start fresh, if needed.
make clean

# Make sure everything compiles.
# This can be skipped if you are just writing tests.
make

# Make sure all tests pass.
make testfast

# Make sure nothing leaks.
# This can be skipped in favor of ‘make testeverything‘, but is faster.
make testfull

# Make sure there are no leaks in ColrC or the many examples.
# This is only needed when you think you’re done with your work,
# and you’d like to commit/push your changes.
make testeverything

# Rebuild the documentation if anything has changed.
make docs
```

If one of them fails, start over. If all of them pass, congratulations. You didn't break anything.

All make targets can be listed with `make help` or `make targets`. I've listed the main targets here.

#### 0.2.4.2 Build

- `make clean`
  - Remove any object files or binaries to force a fresh build.
- `make`
  - Simple running `make` in the source directory will build the `colrc` tool in debug mode.
- `make release`
  - Build a non-debug build for the `colrc` tool.

#### 0.2.4.3 Test

- `make test`
  - Build and run the tests using the address sanitizer options (slowest build time).
- `make testfast`
  - Build and run the tests in debug mode (fastest build time).
- `make testmemcheck`
  - Build and run the tests in debug mode, through `memcheck`.
- `make testfull`
  - Build and run the test in debug mode, in `memcheck` mode, and in "sanitized" mode.
- `make testeverything`
  - Like `make testfull`, but also runs `memcheck` on all source examples, example files, and any examples in the main README. It also builds the coverage report.
- `make testcoverage`
  - Build a coverage report for the tests.
- `make testcoverageview`
  - Open the coverage report in a browser.
- `make cppcheckreport`
  - Build a cppcheck report.
- `make cppcheckview`
  - Open the cppcheck report in a browser.

#### 0.2.4.4 Document

- `make docshtml`
  - Build the HTML documentation. This is faster if you're tweaking the format or looking for mistakes.
- `make docs`
  - Build all documentation (HTML, PDF, GitHub README, etc.)
- `make cleandocs`
  - Remove all generated doc files, to start fresh.

#### 0.2.4.5 Examples

- `make examples`
  - Build all examples in `./examples`. This is not required, but is useful if you've written a new example and you would like to make sure it compiles.
- `make cleanexamples`
  - Remove all example objects/binaries, to start fresh.

### 0.2.5 Tools

#### 0.2.5.1 ColrC Tools

There are several scripts/tools in the `./tools` directory that aid in development. Some of them were created specifically for `make` targets, and some are used for inspecting the state of ColrC. All of them can be used as standalone commands, and all of them support the `-h`/`--help` options.

- `examples/run_examples.sh`
  - Run examples, and run `memcheck` on the examples.
- `test/run_tests.sh`
  - Run tests, `memcheck` examples and the `colrc` tool.
- `clean.sh`
  - Implements the `clean` make targets.
- `cppcheck_errors.py`
  - Lists all possible `cppcheck` errors/warnings, with filtering options.
- `cppcheck_run.py`
  - Run `cppcheck`, generate HTML reports for ColrC.
- `find_python.sh`
  - Locate and report a specific python executable by version.
- `gen_coverage_html.sh`
  - Uses `lcov` to generate an HTML coverage report for ColrC.
- `gen_latex_pdf.sh`
  - Generates the PDF manual from Doxygen's LaTeX output.
- `get_version.sh`
  - Report the current ColrC version (based on the source files).
- `install.sh`
  - Installs and uninstalls the `colrc` executable.
- `is_build.sh`
  - Determines the current build type for `colrc` and `test_colrc` (`debug`, `release`, `sanitize`).
- `make_dist.sh`
  - Creates a small downloadable package for users of ColrC.
- `make_help_fmter.py`
  - Colorizes and formats output for the `make help` target.
- `refactor.sh`
  - Basic refactoring tool, with preview of changes to be made.
- `replacestr.py`

  - Replaces strings in files, with options to preview the changes. `refactor.sh` is implemented with this.

- `snippet.py`
  - Compile and run arbitrary C code, ColrC source examples, ColrC snippets (snippets of C that use ColrC features), with options for running code through `memcheck`, gdb/kdbg, or user-specified tools.
- `undoxy_markdown.py`
  - Generates a GitHub-friendly README from `index.md` for ColrC.
- `unused.py`
  - Display unused and untested functions/macros in the ColrC source.
- `valgrind_run.sh`
  - Runs `colrc` or the tests through `cachegrind`, `callgrind`, or `memcheck`.

If you would like to see the acceptable options or usage strings for these commands, run `<command> -h`.

### 0.2.6 Examples

#### 0.2.6.1 ColrC Examples

The example programs listed here in the documentation exist to show people how to do things in ColrC. They are meant to be brief example programs that showcase a certain ColrC feature. They are automatically compiled and tested for memory leaks when you run the "everything test". There is a `makefile` in the `./examples` directory that knows how to compile all of the example programs by name. Each one can run as a standalone program.

There is a BASH script (`./examples/run_examples.sh`) that will run these example programs with options for filtering by name, running `memcheck` on them, or using the binary name as an argument to another program (gdb/kdbg).

Here are a few of the most common uses for `run_examples.sh`:

```
# Run all examples.
./run_examples.sh

# Run all colr_replace* examples.
./run_examples.sh colr_replace

# Run examples through Valgrind's `memcheck`.
./run_examples.sh simple_example -m

# Run examples through `memcheck`, but only show errors/leaks.
./run_examples.sh simple_example -m -q

# Debug an example using KDbg.
./run_examples.sh simple_example -r kdbg

# Send arguments to KDbg for the example program to use.
# This is like calling `kdbg simple_example -a hello`, which debugs `simple_example hello`.
./run_examples.sh simple_example -r kdbg -- -a hello
```

You can also compile/run all examples from the source directory with a `make` target:

```
# Compile examples that have changed.
make examples

# Run all examples.
make runexamples
```

All of the main features in ColrC should have an example that showcases their usage. If you think of any missing examples, please send an issue or pull-request.

### 0.2.7 Compatibility

#### 0.2.7.1 About

ColrC was written with Linux in mind, specifically Debian-based distributions. If it works on any other system, it is purely by accident and I would like to hear what you're running it on.

#### 0.2.7.2 Porting

If ColrC needs a litle tweak here or there to make it work on your system, please create an issue or a pull-request to let me know. It would be great for ColrC to work on as many machines as possible, but I don't have the resources to test against them all.

#### 0.2.7.3 Windows

Work may be done in the future to make ColrC run on Windows 10+ machines (like Colr.py), but as of right now it is not possible. Again, if you would like to see that happen please create an issue or a pull-request.

## 0.3 Downloads

### 0.3.1 Downloadable Files

Here are a couple downloadable packages from **ColrC**.

#### 0.3.1.1 PDF

This documentation is available in a PDF:

- `ColrC-manual.pdf`

#### 0.3.1.2 Source Files

The **ColrC** header and source file can be downloaded if you don't want to clone the github repo:

- Source Package

## 0.4 Tool

### 0.4.1 About

The ColrC repo includes the **ColrC Tool**, which is a program that colorizes text from the command line. It offers all of the important features from the original colr tool, but operates *much* faster because it was written in a compiled language. You can have both of these installed at the same time. The ColrC version is known as `colrc`, where the original is known as `colr`.

If you would like to use the ColrC tool, you will have to build it and install it.

The ColrC tool can be used in shell scripts or as a standalone application in a variety of ways. Long options are used in the examples, but they all have a single-letter short form as well:

#### 0.4.1.1 Colorizing Text

The most basic use of `colrc` is to colorize text (from arguments or `stdin`). The `FORE`, `BACK`, and `STYLE` arguments are optional, and order only matters when you're not using the explicit `--fore`, `--back`, and `--style` flags.

For instance, creating some red text is as simple as:

```
colrc "Hello World" red
```

If you want to colorize output from another program, use `-` as the text:

```
date | colrc - red
```

If you only want to set the back color or style you would need to be explicit:

```
# Set only the back color, to white:
colrc "Hello World" --back white

# Set only the style, to underline:
colrc "Hello World" --style underline
```

#### 0.4.1.2 Rainbows

The Colr tool can make "rainbowized" text, much like `lolcat` except faster (only because of the language choice).

The options for ColrC do not match `lolcat` exactly, but if you would like to "rainbowize" some text, all you have to do is set the fore or back color to `rainbow`:

```
colrc "Hello World" rainbow
```

One of the most common uses is to pipe some output to ColrC to make it prettier:

```
# "Display a rainbow cookie."
fortune | colrc - rainbow
```

You can also "rainbowize" the background, and optionally set the fore color and style at the same time:

```
# Just the background:
fortune | colrc - --back rainbow

# Fix the foreground and style so the words are more visible:
fortune | colrc - black rainbow bright
```

#### 0.4.1.3 Stripping Colorized Output

If you have a program that doesn't have a `--color=never` or `--nocolor` option, and you'd like to remove all escape-codes from it's output, use `colrc` to strip them.

Using the section above as an example, I'll run `fortune` through `lolcat` and then "undo" all of those fancy colors:

```
fortune | lolcat | colrc --stripcodes
```

The result is like running `fortune` by itself. No colors.

#### 0.4.1.4 Inspecting Colorized Output

The ColrC tool can parse output from another program and list all colors/styles that are found with an example, a name, and the string that produced them:

```
# Have to use -f with lolcat to force colorized output, for this example.
fortune | lolcat -f | colrc --listcodes
```

If that was too much information (too many codes), you can trim the output by listing only *unique* codes:

```
# Again, using -f to force colorized output from lolcat.
fortune | lolcat -f | colrc --listcodes --unique
```

#### 0.4.1.5 Translating Color Codes

ColrC will translate any valid color name (BasicValue), 256-color value (ExtendedValue), RGB value, or Hex color. A "closest match" will be used for basic names and 256-color values when converting to/from RGB and Hex colors.

```
colrc -t red

# Or:
echo "red" | colrc -t
```

To get the closest matching color from an RGB value (for terminals that don't support them):

```
colrc -t '96;96;96'
```

Same thing with hex values:

```
colrc -t '#606060'
```

You'll notice that when you reverse the translation, you get a different RGB/Hex value:

```
# 59 was the closest match from the previous runs.
colrc -t 59
```

### 0.4.2 Tool Building

#### 0.4.2.1 Build

To use the ColrC tool you will have to build it first. A `makefile` is provided, so the actual building only takes one command. Make sure you have all of the `system dependencies` first.

Clone the repo, if you haven't already:

```
git clone https://github.com/welbornprod/colrc.git
```

Make sure you're in the ColrC project directory:

```
cd colrc
```

Finally, run the `make` target:

```
make release
```

The build process doesn't take very long, and when it's done there will be a `colrc` executable in the project directory.

#### 0.4.2.2 Install

Installing is just copying or symlinking the executable somewhere in `$PATH`. There is a `make` target that will let you choose an install path, and do the rest for you:

```
make install

# Install as a symlink instead of a copy:
make installlink
```

By default, it will ask for confirmation before installing or overwriting anything.

#### 0.4.2.3 Uninstall

If `colrc` was installed somewhere in `$PATH`, you can simply run the install script with `--uninstall`, or just:

```
make uninstall
```

## 0.5 File Index

### 0.5.1 File List

Here is a list of all documented files with brief descriptions:



## 0.6 File Documentation

### 0.6.1 colr.c File Reference

Implements everything in the colr.h header.

```
#include "colr.h"
```

Functions

- void _colr_free (void *p)

  *Calls Colr *_free() functions for Colr objects, otherwise just calls free().*
- bool _colr_is_last_arg (void *p)

  *Determines if a void pointer is _ColrLastArg (the last-arg-marker).*
- char * _colr_join (void *joinerp,...)

  *Joins ColorArgs, ColorTexts, and strings (char*) into one long string separated by it's first argument.*
- size_t _colr_join_array_length (void *ps)

  *Determine the length of a NULL-terminated array of strings (char*), ColorArgs, ColorResults, or ColorTexts.*
- size_t _colr_join_arrayn_size (void *joinerp, void *ps, size_t count)

  *Get the size in bytes needed to join an array of strings (char*), ColorArgs, ColorResults, or ColorTexts by another string (char*), ColorArg, ColorResult, or ColorText.*
- size_t _colr_join_size (void *joinerp, va_list args)

  *Parse arguments, just as in _colr_join(), but only return the size needed to allocate the resulting string.*
- size_t _colr_ptr_length (void *p)

  *Get the size, in bytes, needed to convert a ColorArg, ColorResult, ColorText, or string (char*) into a string.*
- char * _colr_ptr_repr (void *p)

  *Determine what kind of pointer is being passed, and call the appropriate <type>_repr function to obtain an allocated string representation.*
- char * _colr_ptr_to_str (void *p)

  *Determine what kind of pointer is being passed, and call the appropriate <type>_to_str function to obtain an allocated string.*
- char * _rainbow (RGB_fmter fmter, const char *s, double freq, size_t offset, size_t spread)

  *Handles multibyte character string (char*) conversion and character iteration for all of the rainbow_functions.*
- bool ArgType_eq (ArgType a, ArgType b)

  *Compares two ArgTypes.*
- char * ArgType_repr (ArgType type)

  *Creates a string (char*) representation of a ArgType.*
- char * ArgType_to_str (ArgType type)

  *Creates a human-friendly string (char*) from an ArgType.*
- bool BasicValue_eq (BasicValue a, BasicValue b)

  *Compares two BasicValues.*
- BasicValue BasicValue_from_esc (const char *s)

  *Convert an escape-code string (char*) to an actual BasicValue enum value.*
- BasicValue BasicValue_from_str (const char *arg)

  *Convert named argument to an actual BasicValue enum value.*
- bool BasicValue_is_invalid (BasicValue bval)

  *Determines whether a BasicValue is invalid.*
- bool BasicValue_is_valid (BasicValue bval)

  *Determines whether a BasicValue is valid.*
- char * BasicValue_repr (BasicValue bval)

  *Creates a string (char*) representation of a BasicValue.*
- int BasicValue_to_ansi (ArgType type, BasicValue bval)

  *Converts a fore/back BasicValue to the actual ansi code number.*
- char * BasicValue_to_str (BasicValue bval)

  *Create a human-friendly string (char*) representation for a BasicValue.*
- ColorArg ColorArg_empty (void)

  *Create a ColorArg with ARGTYPE_NONE and ColorValue.type.TYPE_NONE.*
- bool ColorArg_eq (ColorArg a, ColorArg b)

*Compares two ColorArg structs.*

- char * ColorArg_example (ColorArg carg, bool colorized)

  *Create a string (char*) representation of a ColorArg with a stylized type/name using escape codes built from the ColorArg's values.*
- void ColorArg_free (ColorArg *p)

  *Free allocated memory for a ColorArg.*
- ColorArg ColorArg_from_BasicValue (ArgType type, BasicValue value)

  *Explicit version of ColorArg_from_value that only handles BasicValues.*
- ColorArg ColorArg_from_esc (const char *s)

  *Parse an escape-code string (char*) into a ColorArg.*
- ColorArg ColorArg_from_ExtendedValue (ArgType type, ExtendedValue value)

  *Explicit version of ColorArg_from_value that only handles ExtendedValues.*
- ColorArg ColorArg_from_RGB (ArgType type, RGB value)

  *Explicit version of ColorArg_from_value that only handles RGB structs.*
- ColorArg ColorArg_from_str (ArgType type, const char *colorname)

  *Build a ColorArg (fore, back, or style value) from a known color name/style.*
- ColorArg ColorArg_from_StyleValue (ArgType type, StyleValue value)

  *Explicit version of ColorArg_from_value that only handles StyleValues.*
- ColorArg ColorArg_from_value (ArgType type, ColorType colrtype, void *p)

  *Used with the color_arg macro to dynamically create a ColorArg based on it's argument type.*
- bool ColorArg_is_empty (ColorArg carg)

  *Checks to see if a ColorArg is an empty placeholder.*
- bool ColorArg_is_invalid (ColorArg carg)

  *Checks to see if a ColorArg holds an invalid value.*
- bool ColorArg_is_ptr (void *p)

  *Checks a void pointer to see if it contains a ColorArg struct.*
- bool ColorArg_is_valid (ColorArg carg)

  *Checks to see if a ColorArg holds a valid value.*
- size_t ColorArg_length (ColorArg carg)

  *Returns the length in bytes needed to allocate a string (char*) built with ColorArg_to_esc().*
- char * ColorArg_repr (ColorArg carg)

  *Creates a string (char*) representation for a ColorArg.*
- char * ColorArg_to_esc (ColorArg carg)

  *Converts a ColorArg into an escape code string (char*).*
- bool ColorArg_to_esc_s (char *dest, ColorArg carg)

  *Converts a ColorArg into an escape code string (char*) and fills the destination string.*
- ColorArg * ColorArg_to_ptr (ColorArg carg)

  *Copies a ColorArg into memory and returns the pointer.*
- void ColorArgs_array_free (ColorArg **ps)

  *Free an allocated array of ColorArgs, including the array itself.*
- char * ColorArgs_array_repr (ColorArg **lst)

  *Creates a string representation for an array of ColorArg pointers.*
- ColorArg ** ColorArgs_from_str (const char *s, bool unique)

  *Create an array of ColorArgs from escape-codes found in a string (char*).*
- ColorJustify ColorJustify_empty (void)

  *Creates an "empty" ColorJustify, with JUST_NONE set.*
- bool ColorJustify_eq (ColorJustify a, ColorJustify b)

  *Compares two ColorJustify structs.*
- bool ColorJustify_is_empty (ColorJustify cjust)

  *Checks to see if a ColorJustify is "empty".*
- ColorJustify ColorJustify_new (ColorJustifyMethod method, int width, char padchar)

*Creates a ColorJustify.*

- char * ColorJustify_repr (ColorJustify cjust)

  *Creates a string (char*) representation for a ColorJustify.*
- char * ColorJustifyMethod_repr (ColorJustifyMethod meth)

  *Creates a string (char*) representation for a ColorJustifyMethod.*
- ColorResult ColorResult_empty (void)

  *Creates a ColorResult with .result=NULL and .length=-1, with the appropriate struct marker.*
- bool ColorResult_eq (ColorResult a, ColorResult b)

  *Compares two ColorResults.*
- void ColorResult_free (ColorResult *p)

  *Free allocated memory for a ColorResult and it's .result member.*
- bool ColorResult_is_ptr (void *p)

  *Checks a void pointer to see if it contains a ColorResult struct.*
- size_t ColorResult_length (ColorResult cres)

  *Return the length in bytes (including the null-terminator), that is needed to store the return from ColorResult_to_str() (.result).*
- ColorResult ColorResult_new (char *s)

  *Initialize a new ColorResult with an allocated string (char*).*
- char * ColorResult_repr (ColorResult cres)

  *Create a string representation for a ColorResult.*
- ColorResult * ColorResult_to_ptr (ColorResult cres)

  *Allocate memory for a ColorResult, fill it, and return it.*
- char * ColorResult_to_str (ColorResult cres)

  *Convert a ColorResult into a string (char*).*
- ColorText ColorText_empty (void)

  *Creates an "empty" ColorText with pointers set to NULL.*
- void ColorText_free (ColorText *p)

  *Frees a ColorText and it's ColorArgs.*
- void ColorText_free_args (ColorText *p)

  *Frees the ColorArg members of a ColorText.*
- ColorText ColorText_from_values (char *text,...)

  *Builds a ColorText from 1 mandatory string (char*), and optional fore, back, and style args (pointers to ColorArgs).*
- bool ColorText_has_arg (ColorText ctext, ColorArg carg)

  *Checks to see if a ColorText has a certain ColorArg value set.*
- bool ColorText_has_args (ColorText ctext)

  *Checks to see if a ColorText has any argument values set.*
- bool ColorText_is_empty (ColorText ctext)

  *Checks to see if a ColorText has no usable values.*
- bool ColorText_is_ptr (void *p)

  *Checks a void pointer to see if it contains a ColorText struct.*
- size_t ColorText_length (ColorText ctext)

  *Returns the length in bytes needed to allocate a string (char*) built with ColorText_to_str() with the current text, fore, back, and style members.*
- char * ColorText_repr (ColorText ctext)

  *Allocate a string (char*) representation for a ColorText.*
- ColorText * ColorText_set_just (ColorText *ctext, ColorJustify cjust)

  *Set the ColorJustify method for a ColorText, and return the ColorText.*
- void ColorText_set_values (ColorText *ctext, char *text,...)

  *Initializes an existing ColorText from 1 mandatory string (char*), and optional fore, back, and style args (pointers to ColorArgs).*

- ColorText * ColorText_to_ptr (ColorText ctext)

  *Copies a ColorText into allocated memory and returns the pointer.*
- char * ColorText_to_str (ColorText ctext)

  *Stringifies a ColorText struct, creating a mix of escape codes and text.*
- bool ColorType_eq (ColorType a, ColorType b)

  *Compares two ColorTypes.*
- ColorType ColorType_from_str (const char *arg)

  *Determine which type of color value is desired by name.*
- bool ColorType_is_invalid (ColorType type)

  *Check to see if a ColorType value is considered invalid.*
- bool ColorType_is_valid (ColorType type)

  *Check to see if a ColorType value is considered valid.*
- char * ColorType_repr (ColorType type)

  *Creates a string (char*) representation of a ColorType.*
- char * ColorType_to_str (ColorType type)

  *Create a human-friendly string (char*) representation for a ColorType.*
- ColorValue ColorValue_empty (void)

  *Create an "empty" ColorValue.*
- bool ColorValue_eq (ColorValue a, ColorValue b)

  *Compares two ColorValue structs.*
- char * ColorValue_example (ColorValue cval)

  *Create a string (char*) representation of a ColorValue with a human-friendly type/name.*
- ColorValue ColorValue_from_esc (const char *s)

  *Convert an escape-code string (char*) into a ColorValue.*
- ColorValue ColorValue_from_str (const char *s)

  *Create a ColorValue from a known color name, or RGB string (char*).*
- ColorValue ColorValue_from_value (ColorType type, void *p)

  *Used with the color_val macro to dynamically create a ColorValue based on it's argument type.*
- bool ColorValue_has_BasicValue (ColorValue cval, BasicValue bval)

  *Checks to see if a ColorValue has a BasicValue set.*
- bool ColorValue_has_ExtendedValue (ColorValue cval, ExtendedValue eval)

  *Checks to see if a ColorValue has a ExtendedValue set.*
- bool ColorValue_has_RGB (ColorValue cval, RGB rgb)

  *Checks to see if a ColorValue has a RGB value set.*
- bool ColorValue_has_StyleValue (ColorValue cval, StyleValue sval)

  *Checks to see if a ColorValue has a StyleValue set.*
- bool ColorValue_is_empty (ColorValue cval)

  *Checks to see if a ColorValue is an empty placeholder.*
- bool ColorValue_is_invalid (ColorValue cval)

  *Checks to see if a ColorValue holds an invalid value.*
- bool ColorValue_is_valid (ColorValue cval)

  *Checks to see if a ColorValue holds a valid value.*
- size_t ColorValue_length (ArgType type, ColorValue cval)

  *Returns the length in bytes needed to allocate a string (char*) built with ColorValue_to_esc() with the specified ArgType and ColorValue.*
- char * ColorValue_repr (ColorValue cval)

  *Creates a string (char*) representation of a ColorValue.*
- char * ColorValue_to_esc (ArgType type, ColorValue cval)

  *Converts a ColorValue into an escape code string (char*).*
- bool ColorValue_to_esc_s (char *dest, ArgType type, ColorValue cval)

  *Converts a ColorValue into an escape code string (char*) and fills the destination string.*

- size_t colr_str_code_count (const char *s)
  *Return the number of escape-codes in a string (char*).*
- size_t colr_str_code_len (const char *s)
  *Return the number of bytes that make up all the escape-codes in a string (char*).*
- char * colr_str_copy (char *restrict dest, const char *restrict src, size_t length)
  *Copies a string (char*) like strncpy, but ensures null-termination.*
- bool colr_str_ends_with (const char *restrict s, const char *restrict suffix)
  *Determine if one string (char*) ends with another.*
- char ** colr_str_get_codes (const char *s, bool unique)
  *Get an array of escape-codes from a string (char*).*
- bool colr_str_has_codes (const char *s)
  *Determines if a string (char*) has ANSI escape codes in it.*
- ColrHash colr_str_hash (const char *s)
  *Hash a string using djb2.*
- bool colr_str_is_all (const char *s, const char c)
  *Determines whether a string (char*) consists of only one character, possibly repeated.*
- bool colr_str_is_codes (const char *s)
  *Determines if a string (char*) is composed entirely of escape codes.*
- bool colr_str_is_digits (const char *s)
  *Determines whether all characters in a string (char*) are digits.*
- char * colr_str_ljust (const char *s, int width, const char padchar)
  *Left-justifies a string (char*), ignoring escape codes when measuring the width.*
- void colr_str_lower (char *s)
  *Converts a string (char*) into lower case in place.*
- size_t colr_str_lstrip (char *restrict dest, const char *restrict s, size_t length, const char c)
  *Strip a leading character from a string (char*), filling another string (char*) with the result.*
- char * colr_str_lstrip_char (const char *s, const char c)
  *Strips a leading character from a string (char*), and allocates a new string with the result.*
- char * colr_str_lstrip_chars (const char *restrict s, const char *restrict chars)
  *Removes certain characters from the start of a string (char*) and allocates a new string with the result.*
- size_t colr_str_mb_len (const char *s)
  *Returns the number of characters in a string (char*), taking into account possibly multibyte characters.*
- size_t colr_str_noncode_len (const char *s)
  *Returns the length of string (char*), ignoring escape codes and the the null-terminator.*
- char * colr_str_replace (const char *restrict s, const char *restrict target, const char *restrict repl)
  *Replaces the first substring found in a string (char*).*
- char * colr_str_replace_all (const char *restrict s, const char *restrict target, const char *restrict repl)
  *Replaces the first substring found in a string (char*).*
- char * colr_str_replace_all_ColorArg (const char *restrict s, const char *restrict target, ColorArg *repl)
  *Replace all substrings in a string (char*) with a ColorArg's string result.*
- char * colr_str_replace_all_ColorResult (const char *restrict s, const char *restrict target, ColorResult *repl)
  *Replace all substrings in a string (char*) with a ColorResult's string result.*
- char * colr_str_replace_all_ColorText (const char *restrict s, const char *restrict target, ColorText *repl)
  *Replace all substrings in a string (char*) with a ColorText's string result.*
- char * colr_str_replace_cnt (const char *restrict s, const char *restrict target, const char *restrict repl, int count)

*Replaces one or more substrings in a string (char*).*

- char * colr_str_replace_ColorArg (const char *restrict s, const char *restrict target, ColorArg *repl)

  *Replace a substring in a string (char*) with a ColorArg's string result.*
- char * colr_str_replace_ColorResult (const char *restrict s, const char *restrict target, ColorResult *repl)

  *Replace a substring in a string (char*) with a ColorResult's string result.*
- char * colr_str_replace_ColorText (const char *restrict s, const char *restrict target, ColorText *repl)

  *Replace a substring in a string (char*) with a ColorText's string result.*
- char * colr_str_replace_re (const char *restrict s, const char *restrict pattern, const char *restrict repl, int re_flags)

  *Replaces a substring from a regex pattern string (char*) in a string (char*).*
- char * colr_str_replace_re_all (const char *restrict s, const char *restrict pattern, const char *restrict repl, int re_flags)

  *Replaces all substrings from a regex pattern string (char*) in a string (char*).*
- char * colr_str_replace_re_all_ColorArg (const char *restrict s, const char *restrict pattern, ColorArg *repl, int re_flags)

  *Replace all substrings from a regex pattern string (char*) in a string (char*) with a ColorArg's string result.*
- char * colr_str_replace_re_all_ColorResult (const char *restrict s, const char *restrict pattern, ColorResult *repl, int re_flags)

  *Replace all substrings from a regex pattern string (char*) in a string (char*) with a ColorResult's string result.*
- char * colr_str_replace_re_all_ColorText (const char *restrict s, const char *restrict pattern, ColorText *repl, int re_flags)

  *Replace all substrings from a regex pattern string (char*) in a string (char*) with a ColorText's string result.*
- char * colr_str_replace_re_ColorArg (const char *restrict s, const char *restrict pattern, ColorArg *repl, int re_flags)

  *Replace substrings from a regex pattern string (char*) in a string (char*) with a ColorArg's string result.*
- char * colr_str_replace_re_ColorResult (const char *restrict s, const char *restrict pattern, ColorResult *repl, int re_flags)

  *Replace substrings from a regex pattern string (char*) in a string (char*) with a ColorResult's string result.*
- char * colr_str_replace_re_ColorText (const char *restrict s, const char *restrict pattern, ColorText *repl, int re_flags)

  *Replace substrings from a regex pattern string (char*) in a string (char*) with a ColorText's string result.*
- char * colr_str_replace_re_match (const char *restrict s, regmatch_t *match, const char *restrict repl)

  *Replaces substrings from a single regex match (regmatch_t*) in a string (char*).*
- char * colr_str_replace_re_match_ColorArg (const char *restrict s, regmatch_t *match, ColorArg *repl)

  *Replace substrings from a regex match (regmatch_t*) in a string (char*) with a ColorArg's string result.*
- char * colr_str_replace_re_match_ColorResult (const char *restrict s, regmatch_t *match, ColorResult *repl)

  *Replace substrings from a regex match (regmatch_t*) in a string (char*) with a ColorResult's string result.*
- char * colr_str_replace_re_match_ColorText (const char *restrict s, regmatch_t *match, ColorText *repl)

  *Replace substrings from a regex match (regmatch_t*) in a string (char*) with a ColorText's string result.*
- char * colr_str_replace_re_match_i (const char *restrict ref, char *target, regmatch_t *match, const char *restrict repl)

  *Replaces substrings from a regex match (regmatch_t*) in a string (char*).*

- char * colr_str_replace_re_matches (const char *restrict s, regmatch_t **matches, const char *restrict repl)

  *Replaces substrings from an array of regex match (`regmatch_t*`) in a string (`char*`).*
- char * colr_str_replace_re_matches_ColorArg (const char *restrict s, regmatch_t **matches, ColorArg *repl)

  *Replace substrings from an array of regex matches (`regmatch_t**`) in a string (`char*`) with a ColorArg's string result.*
- char * colr_str_replace_re_matches_ColorResult (const char *restrict s, regmatch_t **matches, ColorResult *repl)

  *Replace substrings from an array of regex matches (`regmatch_t**`) in a string (`char*`) with a ColorResult's string result.*
- char * colr_str_replace_re_matches_ColorText (const char *restrict s, regmatch_t **matches, ColorText *repl)

  *Replace substrings from an array of regex matches (`regmatch_t**`) in a string (`char*`) with a ColorText's string result.*
- char * colr_str_replace_re_pat (const char *restrict s, regex_t *repattern, const char *restrict repl)

  *Replaces regex patterns in a string (`char*`).*
- char * colr_str_replace_re_pat_all (const char *restrict s, regex_t *repattern, const char *restrict repl)

  *Replaces all matches to a regex pattern in a string (`char*`).*
- char * colr_str_replace_re_pat_all_ColorArg (const char *restrict s, regex_t *repattern, ColorArg *repl)

  *Replace all matches to a regex pattern in a string (`char*`) with a ColorArg's string result.*
- char * colr_str_replace_re_pat_all_ColorResult (const char *restrict s, regex_t *repattern, ColorResult *repl)

  *Replace all matches to a regex pattern in a string (`char*`) with a ColorResult's string result.*
- char * colr_str_replace_re_pat_all_ColorText (const char *restrict s, regex_t *repattern, ColorText *repl)

  *Replace all matches to a regex pattern in a string (`char*`) with a ColorText's string result.*
- char * colr_str_replace_re_pat_ColorArg (const char *restrict s, regex_t *repattern, ColorArg *repl)

  *Replace regex patterns in a string (`char*`) with a ColorArg's string result.*
- char * colr_str_replace_re_pat_ColorResult (const char *restrict s, regex_t *repattern, ColorResult *repl)

  *Replace regex patterns in a string (`char*`) with a ColorResult's string result.*
- char * colr_str_replace_re_pat_ColorText (const char *restrict s, regex_t *repattern, ColorText *repl)

  *Replace regex patterns in a string (`char*`) with a ColorText's string result.*
- char * colr_str_repr (const char *s)

  *Convert a string (`char*`) into a representation of a string, by wrapping it in quotes and escaping characters that need escaping.*
- char * colr_str_rjust (const char *s, int width, const char padchar)

  *Right-justifies a string (`char*`), ignoring escape codes when measuring the width.*
- bool colr_str_starts_with (const char *restrict s, const char *restrict prefix)

  *Checks a string (`char*`) for a certain prefix substring.*
- char * colr_str_strip_codes (const char *s)

  *Strips escape codes from a string (`char*`), resulting in a new allocated string.*
- char * colr_str_to_lower (const char *s)

  *Allocate a new lowercase version of a string (`char*`).*
- bool colr_supports_rgb (void)

  *Determine whether the current environment support RGB (True Colors).*
- bool colr_supports_rgb_static (void)

*Same as colr_supports_rgb(), but the environment is only checked on the first call.*

- TermSize colr_term_size (void)

  *Attempts to retrieve the row/column size of the terminal and returns a TermSize.*
- struct winsize colr_win_size (void)

  *Attempts to retrieve a `winsize` struct from an `ioctl` call.*
- struct winsize colr_win_size_env (void)

  *Get window/terminal size using the environment variables `LINES`, `COLUMNS`, or `COLS`.*
- bool ExtendedValue_eq (ExtendedValue a, ExtendedValue b)

  *Compares two ExtendedValues.*
- int ExtendedValue_from_BasicValue (BasicValue bval)

  *Convert a BasicValue into an ExtendedValue.*
- int ExtendedValue_from_esc (const char *s)

  *Convert an escape-code string (`char*`) to an ExtendedValue.*
- int ExtendedValue_from_hex (const char *hexstr)

  *Create an ExtendedValue from a hex string (`char*`).*
- ExtendedValue ExtendedValue_from_hex_default (const char *hexstr, ExtendedValue default_value)

  *Create an ExtendedValue from a hex string (`char*`), but return a default value if the hex string is invalid.*
- ExtendedValue ExtendedValue_from_RGB (RGB rgb)

  *Convert an RGB value into the closest matching ExtendedValue.*
- int ExtendedValue_from_str (const char *arg)

  *Converts a known name, integer string (0-255), or a hex string (`char*`), into an ExtendedValue suitable for the extended-value-based functions.*
- bool ExtendedValue_is_invalid (int eval)

  *Determines whether an integer is an invalid ExtendedValue.*
- bool ExtendedValue_is_valid (int eval)

  *Determines whether an integer is a valid ExtendedValue.*
- char * ExtendedValue_repr (int eval)

  *Creates a string (`char*`) representation of a ExtendedValue.*
- char * ExtendedValue_to_str (ExtendedValue eval)

  *Creates a human-friendly string (`char*`) from an ExtendedValue's actual value, suitable for use with ExtendedValue_from_str().*
- void format_bg (char *out, BasicValue value)

  *Create an escape code for a background color.*
- void format_bg_RGB (char *out, RGB rgb)

  *Create an escape code for a true color (rgb) background color using values from an RGB struct.*
- void format_bg_RGB_term (char *out, RGB rgb)

  *Create an escape code for a true color (rgb) fore color using an RGB struct's values, approximating 256-color values.*
- void format_bgx (char *out, unsigned char num)

  *Create an escape code for an extended background color.*
- void format_fg (char *out, BasicValue value)

  *Create an escape code for a fore color.*
- void format_fg_RGB (char *out, RGB rgb)

  *Create an escape code for a true color (rgb) fore color using an RGB struct's values.*
- void format_fg_RGB_term (char *out, RGB rgb)

  *Create an escape code for a true color (rgb) fore color using an RGB struct's values, approximating 256-color values.*
- void format_fgx (char *out, unsigned char num)

  *Create an escape code for an extended fore color.*
- void format_style (char *out, StyleValue style)

  *Create an escape code for a style.*

- char * rainbow_bg (const char *s, double freq, size_t offset, size_t spread)
  *Rainbow-ize some text using rgb back colors, lolcat style.*
- char * rainbow_bg_term (const char *s, double freq, size_t offset, size_t spread)
  *This is exactly like rainbow_bg(), except it uses colors that are closer to the standard 256-color values.*
- char * rainbow_fg (const char *s, double freq, size_t offset, size_t spread)
  *Rainbow-ize some text using rgb fore colors, lolcat style.*
- char * rainbow_fg_term (const char *s, double freq, size_t offset, size_t spread)
  *This is exactly like rainbow_fg(), except it uses colors that are closer to the standard 256-color values.*
- RGB rainbow_step (double freq, size_t offset)
  *A single step in rainbow-izing produces the next color in the "rainbow" as an RGB value.*
- unsigned char RGB_average (RGB rgb)
  *Return the average for an RGB value.*
- bool RGB_eq (RGB a, RGB b)
  *Compare two RGB structs.*
- RGB RGB_from_BasicValue (BasicValue bval)
  *Return an RGB value from a known BasicValue.*
- int RGB_from_esc (const char *s, RGB *rgb)
  *Convert an escape-code string (char*) to an actual RGB value.*
- RGB RGB_from_ExtendedValue (ExtendedValue eval)
  *Return an RGB value from a known ExtendedValue.*
- int RGB_from_hex (const char *hexstr, RGB *rgb)
  *Convert a hex color into an RGB value.*
- RGB RGB_from_hex_default (const char *hexstr, RGB default_value)
  *Convert a hex color into an RGB value, but use a default value when errors occur.*
- int RGB_from_str (const char *arg, RGB *rgb)
  *Convert an RGB string (char*) into an RGB value.*
- RGB RGB_grayscale (RGB rgb)
  *Return a grayscale version of an RGB value.*
- RGB RGB_inverted (RGB rgb)
  *Make a copy of an RGB value, with the colors "inverted" (like highlighting text in the terminal).*
- RGB RGB_monochrome (RGB rgb)
  *Convert an RGB value into either black or white, depending on it's average grayscale value.*
- char * RGB_repr (RGB rgb)
  *Creates a string (char*) representation for an RGB value.*
- char * RGB_to_hex (RGB rgb)
  *Converts an RGB value into a hex string (char*).*
- char * RGB_to_str (RGB rgb)
  *Convert an RGB value into a human-friendly RGB string (char*) suitable for input to RGB_from_str().*
- RGB RGB_to_term_RGB (RGB rgb)
  *Convert an RGB value into it's nearest terminal-friendly RGB value.*
- bool StyleValue_eq (StyleValue a, StyleValue b)
  *Compares two StyleValues.*
- StyleValue StyleValue_from_esc (const char *s)
  *Convert an escape-code string (char*) to an actual StyleValue enum value.*
- StyleValue StyleValue_from_str (const char *arg)
  *Convert a named argument to actual StyleValue enum value.*
- bool StyleValue_is_invalid (StyleValue sval)
  *Determines whether a StyleValue is invalid.*
- bool StyleValue_is_valid (StyleValue sval)
  *Determines whether a StyleValue is valid.*
- char * StyleValue_repr (StyleValue sval)

*Creates a string (`char*`) representation of a StyleValue.*

- char * StyleValue_to_str (StyleValue sval)

  *Create a human-friendly string (`char*`) representation for a StyleValue.*
- char * TermSize_repr (TermSize ts)

  *Create a string (`char*`) representation for a TermSize.*

Variables

- const BasicInfo basic_names [ ]

  *An array of BasicInfo items, used with BasicValue_from_str().*
- const size_t basic_names_len = sizeof(basic_names) / sizeof(basic_names[0])

  *Length of basic_names.*
- const ColorNameData colr_name_data [ ]

  *An array that holds a known color name, it's ExtendedValue, and it's RGB value.*
- const size_t colr_name_data_len = sizeof(colr_name_data) / sizeof(colr_name_data[0])

  *Length of colr_name_data.*
- int colr_printf_esc_mod = 0

  *Integer to test for the presence of the "escaped output modifier" in colr_printf_handler.*
- const RGB ext2rgb_map [ ]

  *A map from ExtendedValue (256-color) to RGB value, where the index is the is the ExtendedValue, and the value is the RGB.*
- const size_t ext2rgb_map_len = sizeof(ext2rgb_map) / sizeof(ext2rgb_map[0])

  *Length of ext2rgb_map (should always be 256).*
- const ExtendedInfo extended_names [ ]

  *An array of ExtendedInfo, used with ExtendedValue_from_str().*
- const size_t extended_names_len = sizeof(extended_names) / sizeof(extended_names[0])

  *Length of extended_names.*
- const StyleInfo style_names [ ]

  *An array of StyleInfo items, used with StyleName_from_str().*
- const size_t style_names_len = sizeof(style_names) / sizeof(style_names[0])

  *Length of style_names.*

#### 0.6.1.1 Detailed Description

Implements everything in the colr.h header.

#### 0.6.1.2 Function Documentation

##### 0.6.1.2.1 _colr_free()

```
void _colr_free (
            void * p )
```

Calls Colr `*_free()` functions for Colr objects, otherwise just calls `free()`.

You should use the colr_free() macro instead.

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| in | *p* | Pointer to a heap-allocated object. |

#### 0.6.1.2.2 _colr_is_last_arg()

```
bool _colr_is_last_arg (
            void * p )
```

Determines if a void pointer is _ColrLastArg (the last-arg-marker).

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| in | *p* | The pointer to check. |

Returns

`true` if the pointer is _ColrLastArg, otherwise `false`.

#### 0.6.1.2.3 _colr_join()

```
char* _colr_join (
            void * joinerp,
             ...  )
```

Joins ColorArgs, ColorTexts, and strings (`char*`) into one long string separated by it's first argument.

This will free() any `ColorArg`s, `ColorResult`s, or `ColorText`s that are passed in. It is backing the colr_cat(), colr_join(), Colr_cat(), and Colr_join() macros, and enables easy throw-away color values.

Any plain strings that are passed in are left alone. It is up to the caller to free those. ColrC only manages the temporary Colr-based objects needed to build up these strings.

You should use colr_cat(), colr_join(), Colr_cat(), and Colr_join() macros instead.

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| in | *joinerp* | The joiner (any ColorArg, ColorResult, ColorText, or string). |
| in | ... | Zero or more ColorArgs, ColorResults, ColorTexts, or strings to join by the joiner. |

Returns

An allocated string with mixed escape codes/strings. CODE_RESET_ALL is appended to all ColorText arguments. This allows easy part-colored messages.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*. Also, `NULL` will be returned if `joinerp` is `NULL`.

#### 0.6.1.2.4 _colr_join_array_length()

```
size_t _colr_join_array_length (
            void * ps )
```

Determine the length of a `NULL`-terminated array of strings (`char*`), `ColorArg`s, `ColorResult`s, or `ColorText`s.

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| in | *ps* | A `NULL`-terminated array of `ColorArg`s, `ColorResult`s, `ColorText`s, or strings (`char*`). |

Returns

The number of items (before `NULL`) in the array.

Referenced by colr_join_array().

#### 0.6.1.2.5 _colr_join_arrayn_size()

```
size_t _colr_join_arrayn_size (
            void * joinerp,
            void * ps,
            size_t count )
```

Get the size in bytes needed to join an array of strings (`char*`), `ColorArg`s, `ColorResult`s, or `ColorText`s by another string (`char*`), ColorArg, ColorResult, or ColorText.

This is used to allocate memory in the _colr_join_array() function.

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| in | *joinerp* | The joiner (any ColorArg, ColorResult, ColorText, or string). |
| in | *ps* | An array of pointers to ColorArgs, ColorResults, ColorTexts, or strings. The array must have NULL as the last item if count is greater than the total number of items. |
| in | *count* | Total number of items in the array. |

Returns

The number of bytes needed to allocate the result of colr_join_arrayn(), possibly 0.

See also

colr
colr_join
colr_join_array

Referenced by colr_join_arrayn().

#### 0.6.1.2.6 _colr_join_size()

```
size_t _colr_join_size (
            void * joinerp,
            va_list args )
```

Parse arguments, just as in _colr_join(), but only return the size needed to allocate the resulting string.

This allows _colr_join() to allocate once, instead of reallocating for each argument that is passed.

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| in | *joinerp* | The joiner (any ColorArg, ColorText, or string (char*)). |
| in | *args* | A va_list with zero or more ColorArgs, ColorTexts, or strings (char*) to join. |

Returns

The length (in bytes) needed to allocate a string built with _colr_cat(). This function will return 0 if joinerp is NULL/empty). Except for 0, it will never return anything less than CODE_RE↩ SET_LEN.

See also

_colr

Referenced by _colr_join().

#### 0.6.1.2.7 _colr_ptr_length()

```
size_t _colr_ptr_length (
            void * p )
```

Get the size, in bytes, needed to convert a ColorArg, ColorResult, ColorText, or string (`char*`) into a string.

This is used in the variadic _colr* functions.

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| `in` | *p* | A ColorArg pointer, ColorResult pointer, ColorText pointer, or string (`char*`). |

Returns

The length needed to convert the object into a string (`strlen() + 1` for strings).

Referenced by _colr_join_arrayn_size(), and _colr_join_size().

#### 0.6.1.2.8 _colr_ptr_repr()

```
char* _colr_ptr_repr (
            void * p )
```

Determine what kind of pointer is being passed, and call the appropriate <type>_repr function to obtain an allocated string representation.

You should use colr_repr() instead.

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| in | *p* | A ColorArg pointer, ColorResult pointer, ColorText pointer, or string. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

colr_repr

#### 0.6.1.2.9 _colr_ptr_to_str()

```
char* _colr_ptr_to_str (
            void * p )
```

Determine what kind of pointer is being passed, and call the appropriate <type>_to_str function to obtain an allocated string.

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| in | *p* | A ColorArg pointer, ColorResult pointer, ColorText pointer, or string. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

#### 0.6.1.2.10 _rainbow()

```
char* _rainbow (
            RGB_fmter fmter,
            const char * s,
            double freq,
            size_t offset,
            size_t spread )
```

Handles multibyte character string (`char*`) conversion and character iteration for all of the rainbow_ functions.

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| in | *fmter* | A formatter function (RGB_fmter) that can create escape codes from RGB values. |
| in | *s* | The string to "rainbowize".<br>Input *must be `null`-terminated*. |
| in | *freq* | The "tightness" for colors. |
| in | *offset* | The starting offset into the rainbow. |
| in | *spread* | Number of characters per color. |

Returns

An allocated string (`char*`) with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

Referenced by rainbow_bg(), rainbow_bg_term(), rainbow_fg(), and rainbow_fg_term().

#### 0.6.1.2.11 ArgType_eq()

```
bool ArgType_eq (
            ArgType a,
            ArgType b )
```

Compares two ArgTypes.

This is used to implement colr_eq().

Parameters

| | | |
|---|---|---|
| in | *a* | The first ArgType to compare. |
| in | *b* | The second ArgType to compare. |

Returns

`true` if they are equal, otherwise `false`.

#### 0.6.1.2.12 ArgType_repr()

```
char* ArgType_repr (
            ArgType type )
```

Creates a string (`char*`) representation of a ArgType.

Parameters

| | | |
|---|---|---|
| in | *type* | An ArgType to get the type from. |

Returns

A pointer to an allocated string.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

ArgType

Referenced by ColorArg_repr().

#### 0.6.1.2.13 ArgType_to_str()

```
char* ArgType_to_str (
            ArgType type )
```

Creates a human-friendly string (`char*`) from an ArgType.

Parameters

| | | |
|---|---|---|
| in | *type* | An ArgType to get the type from. |

Returns

A pointer to an allocated string.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

ArgType

Referenced by ColorArg_example().

#### 0.6.1.2.14 BasicValue_eq()

```
bool BasicValue_eq (
            BasicValue a,
            BasicValue b )
```

Compares two BasicValues.

This is used to implement colr_eq().

Parameters

| | | |
|---|---|---|
| in | *a* | The first BasicValue to compare. |
| in | *b* | The second BasicValue to compare. |

Returns

`true` if they are equal, otherwise `false`.

See also

BasicValue

#### 0.6.1.2.15 BasicValue_from_esc()

```
BasicValue BasicValue_from_esc (
            const char * s )
```

Convert an escape-code string (`char*`) to an actual BasicValue enum value.

Parameters

| | | |
|---|---|---|
| in | *s* | Escape-code string.<br>*Must be `null`-terminated.* |

Return values

| | |
|---|---|
| *BasicValue* | value on success. |
| *BASIC_INVALID* | on error (or if `s` is `NULL`). |
| *BASIC_INVALID_RANGE* | if the code number was outside of the range `0-255`. |

See also

BasicValue

#### 0.6.1.2.16 BasicValue_from_str()

```
BasicValue BasicValue_from_str (
            const char * arg )
```

Convert named argument to an actual BasicValue enum value.

Parameters

| | | |
|---|---|---|
| in | *arg* | Color name to find the BasicValue for. |

Returns

BasicValue value on success, or BASIC_INVALID on error.

See also

BasicValue

#### 0.6.1.2.17 BasicValue_is_invalid()

```
bool BasicValue_is_invalid (
            BasicValue bval )
```

Determines whether a BasicValue is invalid.

Parameters

| | | |
|---|---|---|
| in | *bval* | A BasicValue to check. |

Returns

`true` if the value is considered invalid, otherwise `false`.

See also

BasicValue

Referenced by ExtendedValue_from_BasicValue().

#### 0.6.1.2.18 BasicValue_is_valid()

```
bool BasicValue_is_valid (
            BasicValue bval )
```

Determines whether a BasicValue is valid.

Parameters

| | | |
|---|---|---|
| in | *bval* | A BasicValue to check. |

Returns

`true` if the value is considered valid, otherwise `false`.

See also

BasicValue

#### 0.6.1.2.19 BasicValue_repr()

```
char* BasicValue_repr (
            BasicValue bval )
```

Creates a string (`char*`) representation of a BasicValue.

Parameters

| | | |
|---|---|---|
| `in` | *bval* | A BasicValue to get the value from. |

Returns

A pointer to an allocated string.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

BasicValue

#### 0.6.1.2.20 BasicValue_to_ansi()

```
int BasicValue_to_ansi (
            ArgType type,
            BasicValue bval )
```

Converts a fore/back BasicValue to the actual ansi code number.

Parameters

| | | |
|---|---|---|
| `in` | *type* | ArgType (FORE/BACK). |
| `in` | *bval* | BasicValue to convert. |

Returns

An integer usable with basic escape code fore/back colors.

See also

BasicValue

Referenced by format_bg(), and format_fg().

#### 0.6.1.2.21 BasicValue_to_str()

```
char* BasicValue_to_str (
            BasicValue bval )
```

Create a human-friendly string (`char*`) representation for a BasicValue.

Parameters

| | | |
|---|---|---|
| `in` | *bval* | BasicValue to get the name for. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

BasicValue

#### 0.6.1.2.22 ColorArg_empty()

```
ColorArg ColorArg_empty (
            void  )
```

Create a ColorArg with ARGTYPE_NONE and ColorValue.type.TYPE_NONE.

This is used to pass "empty" fore/back/style args to the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros, where `NULL` may have a different meaning for users of the ColorArg.

Returns

`(ColorArg){.type=ARGTYPE_NONE, .value.type=TYPE_NONE}`

See also

ColorArg_is_empty
ColorValue_empty

#### 0.6.1.2.23 ColorArg_eq()

```
bool ColorArg_eq (
            ColorArg a,
            ColorArg b )
```

Compares two ColorArg structs.

They are considered "equal" if their `.type` and `.value` match.

Parameters

| | | |
|---|---|---|
| in | *a* | First ColorArg to compare. |
| in | *b* | Second ColorArg to compare. |

Returns

`true` if they are equal, otherwise `false`.

See also

ColorArg

Referenced by ColorText_has_arg().

#### 0.6.1.2.24 ColorArg_example()

```
char* ColorArg_example (
            ColorArg carg,
            bool colorized )
```

Create a string (`char*`) representation of a ColorArg with a stylized type/name using escape codes built from the ColorArg's values.

Parameters

| | | |
|---|---|---|
| in | *carg* | A ColorArg to get an example string for. |
| in | *colorized* | Whether to include a colorized example. If set to `false`, there will be no escape-codes in the string. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

ColorArg

#### 0.6.1.2.25 ColorArg_free()

```
void ColorArg_free (
            ColorArg * p )
```

Free allocated memory for a ColorArg.

This has no advantage over `free(colorarg)` right now, it is used in debugging, and may be extended in the future. It's better just to use it (or the colr_free() macro).

Parameters

| | | |
|---|---|---|
| in | *p* | ColorArg to free. |

See also

ColorArg

Referenced by _colr_free(), _colr_join(), ColorText_free_args(), colr_printf_handler(), colr_str_replace_all_ColorArg(), colr_str_replace_ColorArg(), colr_str_replace_re_all_ColorArg(), colr_str_replace_re_ColorArg(), colr_str_replace_re_match_ColorArg(), colr_str_replace_re_matches_ColorArg(), colr_str_replace_re_pat_all_ColorArg(), and colr_str_replace_re_pat_ColorArg().

#### 0.6.1.2.26 ColorArg_from_BasicValue()

```
ColorArg ColorArg_from_BasicValue (
            ArgType type,
            BasicValue value )
```

Explicit version of ColorArg_from_value that only handles BasicValues.

This is used in some macros to aid in dynamic escape code creation.

Parameters

| | | |
|---|---|---|
| in | *type* | ArgType (FORE, BACK, STYLE). |
| in | *value* | BasicValue to use. |

Returns

A ColorArg, with the `.value.type` member possibly set to `TYPE_INVALID`.

See also

ColorArg

#### 0.6.1.2.27 ColorArg_from_esc()

```
ColorArg ColorArg_from_esc (
            const char * s )
```

Parse an escape-code string (`char*`) into a ColorArg.

For malformed escape-codes the `.type` member will be `ARGTYPE_NONE`, and the `.value.type` member will be set to `TYPE_INVALID`. This means that `ColorArg_is_invalid(carg) == true`.

Parameters

| | | |
|---|---|---|
| in | *s* | The escape code to parse. It must not have extra characters. |

Returns

An initialized ColorArg, possibly invalid.

See also

ColorArg
colr_str_get_codes
ColorValue_from_esc
BasicValue_from_esc
ExtendedValue_from_esc
StyleValue_from_esc
RGB_from_esc

Referenced by ColorArgs_from_str().

#### 0.6.1.2.28 ColorArg_from_ExtendedValue()

```
ColorArg ColorArg_from_ExtendedValue (
            ArgType type,
            ExtendedValue value )
```

Explicit version of ColorArg_from_value that only handles ExtendedValues.

This is used in some macros to aid in dynamic escape code creation.

Parameters

| | | |
|---|---|---|
| in | *type* | ArgType (FORE, BACK, STYLE). |
| in | *value* | ExtendedValue to use. |

Returns

A ColorArg, with the `.value.type` member possibly set to `TYPE_INVALID`.

See also

ColorArg

#### 0.6.1.2.29 ColorArg_from_RGB()

```
ColorArg ColorArg_from_RGB (
            ArgType type,
            RGB value )
```

Explicit version of ColorArg_from_value that only handles RGB structs.

This is used in some macros to aid in dynamic escape code creation.

Parameters

| | | |
|---|---|---|
| in | *type* | ArgType (FORE, BACK, STYLE). |
| in | *value* | RGB struct to use. |

Returns

A ColorArg, with the `.value.type` member possibly set to `TYPE_INVALID`.

See also

ColorArg

#### 0.6.1.2.30 ColorArg_from_str()

```
ColorArg ColorArg_from_str (
            ArgType type,
            const char * colorname )
```

Build a ColorArg (fore, back, or style value) from a known color name/style.

The `.value.type` attribute can be checked for an invalid type, or you can call ColorArg_is_↩ invalid(x).

Parameters

| | | |
|---|---|---|
| in | *type* | ArgType (FORE, BACK, STYLE). |
| in | *colorname* | A known color name/style. |

Returns

A ColorArg struct with usable values.

See also

ColorArg

#### 0.6.1.2.31 ColorArg_from_StyleValue()

```
ColorArg ColorArg_from_StyleValue (
            ArgType type,
            StyleValue value )
```

Explicit version of ColorArg_from_value that only handles StyleValues.

This is used in some macros to aid in dynamic escape code creation.

Parameters

| | | |
|---|---|---|
| in | *type* | ArgType (FORE, BACK, STYLE). |
| in | *value* | StyleValue to use. |

Returns

A ColorArg, with the `.value.type` member possibly set to `TYPE_INVALID`.

See also

ColorArg

#### 0.6.1.2.32 ColorArg_from_value()

```
ColorArg ColorArg_from_value (
            ArgType type,
            ColorType colrtype,
            void * p )
```

Used with the color_arg macro to dynamically create a ColorArg based on it's argument type.

Parameters

| | | |
|---|---|---|
| in | *type* | ArgType value, to mark the type of ColorArg. |
| in | *colrtype* | ColorType value, to mark the type of ColorValue. |
| in | *p* | A pointer to either a BasicValue, ExtendedValue, or a RGB. |

Returns

A ColorArg struct with the appropriate `.value.type` member set for the value that was passed. For invalid types the `.value.type` member may be set to one of:

- TYPE_INVALID
- TYPE_INVALID_EXT_RANGE
- TYPE_INVALID_RGB_RANGE

See also

ColorArg

#### 0.6.1.2.33 ColorArg_is_empty()

```
bool ColorArg_is_empty (
            ColorArg carg )
```

Checks to see if a ColorArg is an empty placeholder.

A ColorArg is empty if it's `.type` is set to `ARGTYPE_NONE`.

Parameters

| | | |
|---|---|---|
| `in` | *carg* | A ColorArg to check. |

Returns

`true` if the ColorArg is considered "empty", otherwise `false`.

Referenced by ColorArg_length(), ColorArg_to_esc(), ColorArg_to_esc_s(), ColorText_has_args(), and ColorText_to_str().

#### 0.6.1.2.34 ColorArg_is_invalid()

```
bool ColorArg_is_invalid (
            ColorArg carg )
```

Checks to see if a ColorArg holds an invalid value.

Parameters

| | | |
|---|---|---|
| `in` | *carg* | ColorArg struct to check. |

Returns

`true` if the value is invalid, otherwise `false`.

See also

ColorArg

#### 0.6.1.2.35 ColorArg_is_ptr()

```
bool ColorArg_is_ptr (
            void * p )
```

Checks a void pointer to see if it contains a ColorArg struct.

The first member of a ColorArg is a marker.

Parameters

| | | |
|---|---|---|
| in | *p* | A void pointer to check. |

Returns

`true` if the pointer is a ColorArg, otherwise `false`.

See also

ColorArg

Referenced by _colr_free(), _colr_join(), _colr_join_array_length(), _colr_join_arrayn_size(), _colr_ptr_length(), _colr_ptr_repr(), _colr_ptr_to_str(), ColorText_from_values(), ColorText_set_values(), colr_join_arrayn(), and colr_printf_handler().

#### 0.6.1.2.36 ColorArg_is_valid()

```
bool ColorArg_is_valid (
            ColorArg carg )
```

Checks to see if a ColorArg holds a valid value.

Parameters

| | | |
|---|---|---|
| in | *carg* | ColorArg struct to check. |

Returns

`true` if the value is valid, otherwise `false`.

See also

ColorArg

#### 0.6.1.2.37 ColorArg_length()

```
size_t ColorArg_length (
            ColorArg carg )
```

Returns the length in bytes needed to allocate a string (`char*`) built with ColorArg_to_esc().

Parameters

| | | |
|---|---|---|
| in | *carg* | ColorArg to use. |

Returns

The length (`size_t`) needed to allocate a ColorArg's string, or `1` (size of an empty string) for invalid/empty arg types/values.

See also

ColorArg

Referenced by _colr_join_arrayn_size(), _colr_ptr_length(), and ColorText_length().

#### 0.6.1.2.38 ColorArg_repr()

```
char* ColorArg_repr (
            ColorArg carg )
```

Creates a string (`char*`) representation for a ColorArg.

Allocates memory for the string representation.

Parameters

| | | |
|---|---|---|
| in | *carg* | ColorArg struct to get the representation for. |

Returns

Allocated string for the representation.
*You must `free()` the memory allocated by this function.*

See also

ColorArg

Referenced by _colr_ptr_repr(), and ColorText_repr().

#### 0.6.1.2.39 ColorArg_to_esc()

```
char* ColorArg_to_esc (
            ColorArg carg )
```

Converts a ColorArg into an escape code string (`char*`).

Allocates memory for the string.

If the ColorArg is empty (`ARGTYPE_NONE`), an empty string is returned.

If the ColorValue is invalid, an empty string is returned. You must still free the empty string.

Parameters

| | | |
|---|---|---|
| `in` | *carg* | ColorArg to get the ArgType and ColorValue from. |

Returns

Allocated string for the escape code.
*You must `free()` the memory allocated by this function.* If the ColorArg is considered "empty", or the ColorValue is invalid, then `NULL` is returned.

See also

ColorArg

Referenced by _colr_join(), _colr_ptr_to_str(), ColorText_to_str(), colr_join_arrayn(), colr_printf_handler(), colr_str_replace_all_ColorArg(), colr_str_replace_ColorArg(), colr_str_replace_re_all_ColorArg(), colr_str_replace_re_ColorArg(), colr_str_replace_re_match_ColorArg(), colr_str_replace_re_matches_ColorArg(), colr_str_replace_re_pat_all_ColorArg(), and colr_str_replace_re_pat_ColorArg().

#### 0.6.1.2.40 ColorArg_to_esc_s()

```
bool ColorArg_to_esc_s (
            char * dest,
            ColorArg carg )
```

Converts a ColorArg into an escape code string (`char*`) and fills the destination string.

If the ColorArg is empty (`ARGTYPE_NONE`), `dest[0]` is set to "\0".

If the ColorValue is invalid, `dest[0]` is set to "\0".

Parameters

| | | |
|---|---|---|
| `in` | *dest* | Destination for the escape code string. *Must have room for the code type being used.* See ColorArg_length() for determining the size needed. |
| `in` | *carg* | ColorArg to get the ArgType and ColorValue from. |

Returns

`true` if the ColorArg was valid, otherwise `false`.

See also

ColorArg

#### 0.6.1.2.41 ColorArg_to_ptr()

```
ColorArg* ColorArg_to_ptr (
            ColorArg carg )
```

Copies a ColorArg into memory and returns the pointer.

You must free() the memory if you call this directly.

Parameters

| | | |
|---|---|---|
| `in` | *carg* | ColorArg to copy/allocate for. |

Returns

Pointer to a heap-allocated ColorArg.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

ColorArg

Referenced by ColorArgs_from_str().

#### 0.6.1.2.42 ColorArgs_array_free()

```
void ColorArgs_array_free (
            ColorArg ** ps )
```

Free an allocated array of ColorArgs, including the array itself.

Each individual ColorArg will be released, and finally the allocated memory for the array of pointers will be released.

Parameters

| | | |
|---|---|---|
| `in` | *ps* | A pointer to an array of ColorArgs, where `NULL` is the last item. |

#### 0.6.1.2.43 ColorArgs_array_repr()

```
char* ColorArgs_array_repr (
            ColorArg ** lst )
```

Creates a string representation for an array of ColorArg pointers.

Parameters

| | | |
|---|---|---|
| in | *lst* | The ColorArg array to create the representation for (`ColorArg**`). |

Returns

An allocated string, or `NULL` if `lst` is `NULL`, or the allocation fails.

#### 0.6.1.2.44 ColorArgs_from_str()

```
ColorArg** ColorArgs_from_str (
            const char * s,
            bool unique )
```

Create an array of ColorArgs from escape-codes found in a string (`char*`).

This uses ColorArg_from_esc() and colr_str_get_codes() to build a heap-allocated array of heap-allocated ColorArgs.

Parameters

| | | |
|---|---|---|
| in | *s* | A string to get the escape-codes from.<br>*Must be `null`-terminated.* |
| in | *unique* | Whether to only include *unique* ColorArgs. |

Returns

An allocated array of ColorArg pointers, where the last element is `NULL`.
*You must `free()` the memory allocated by this function.*

Return values

| | |
|---|---|
| *If* | `s` is `NULL`, or empty, or there are otherwise no escape-codes found in the string, then `NULL` is returned. |
| *On* | success, there will be at least two pointers behind the return value. The last pointer is always `NULL`. |

#### 0.6.1.2.45 ColorJustify_empty()

```
ColorJustify ColorJustify_empty (
            void  )
```

Creates an "empty" ColorJustify, with JUST_NONE set.

Returns

An initialized ColorJustify, with no justification method set.

See also

ColorJustify

Referenced by ColorText_empty().

#### 0.6.1.2.46 ColorJustify_eq()

```
bool ColorJustify_eq (
            ColorJustify a,
            ColorJustify b )
```

Compares two ColorJustify structs.

They are considered "equal" if their member values match.

Parameters

| | | |
|---|---|---|
| in | *a* | First ColorJustify to compare. |
| in | *b* | Second ColorJustify to compare. |

Returns

`true` if they are equal, otherwise `false`.

See also

ColorJustify

#### 0.6.1.2.47 ColorJustify_is_empty()

```
bool ColorJustify_is_empty (
            ColorJustify cjust )
```

Checks to see if a ColorJustify is "empty".

A ColorJustify is considered "empty" if the `.method` member is set to `JUST_NONE`.

Parameters

| | | |
|---|---|---|
| in | *cjust* | The ColorJustify to check. |

Returns

`true` if the ColorJustify is empty, otherwise `false`.

See also

ColorJustify
ColorJustify_empty

Referenced by ColorText_is_empty(), and ColorText_length().

#### 0.6.1.2.48 ColorJustify_new()

```
ColorJustify ColorJustify_new (
             ColorJustifyMethod method,
             int width,
             char padchar )
```

Creates a ColorJustify.

This is used to ensure every ColorJustify has it's `.marker` member set correctly.

Parameters

| | | |
|---|---|---|
| in | *method* | ColorJustifyMethod to use. |
| in | *width* | Width for justification. If `0` is given, ColorText will use the width from colr_term_size(). |
| in | *padchar* | Padding character to use. If `0` is given, the default, space (" "), is used. |

Returns

An initialized ColorJustify.

#### 0.6.1.2.49 ColorJustify_repr()

```
char* ColorJustify_repr (
             ColorJustify cjust )
```

Creates a string (`char*`) representation for a ColorJustify.

Allocates memory for the string representation.

Parameters

| | | |
|---|---|---|
| in | *cjust* | ColorJustify struct to get the representation for. |

Returns

Allocated string for the representation.
*You must `free()` the memory allocated by this function.*

See also

ColorJustify

Referenced by ColorText_repr().

#### 0.6.1.2.50 ColorJustifyMethod_repr()

```
char* ColorJustifyMethod_repr (
            ColorJustifyMethod meth )
```

Creates a string (`char*`) representation for a ColorJustifyMethod.

Allocates memory for the string representation.

Parameters

| | | |
|---|---|---|
| in | *meth* | ColorJustifyMethod to get the representation for. |

Returns

Allocated string for the representation.
*You must `free()` the memory allocated by this function.*

See also

ColorJustifyMethod

Referenced by ColorJustify_repr().

#### 0.6.1.2.51 ColorResult_empty()

```
ColorResult ColorResult_empty (
            void  )
```

Creates a ColorResult with `.result=NULL` and `.length=-1`, with the appropriate struct marker.

Returns

An "empty" (initialized) ColorResult.

See also

ColorResult

Referenced by ColorResult_new().

#### 0.6.1.2.52 ColorResult_eq()

```
bool ColorResult_eq (
            ColorResult a,
            ColorResult b )
```

Compares two ColorResults.

They are equal if all of their members are equal, excluding the memory address for the `.result` member.

Parameters

| | | |
|---|---|---|
| `in` | *a* | First ColorResult to compare. |
| `in` | *b* | Second ColorResult to compare. |

Returns

`true` if they are equal, otherwise `false`.

See also

ColorResult

#### 0.6.1.2.53 ColorResult_free()

```
void ColorResult_free (
            ColorResult * p )
```

Free allocated memory for a ColorResult and it's `.result` member.

Parameters

| | | |
|---|---|---|
| `in` | *p* | A ColorResult with a `NULL` or heap-allocated `.result` member. |

See also

ColorResult

Referenced by _colr_free(), _colr_join(), colr_printf_handler(), colr_str_replace_all_ColorResult(), colr_str_replace_ColorResult(), colr_str_replace_re_all_ColorResult(), colr_str_replace_re_ColorResult(), colr_str_replace_re_match_ColorResult(), colr_str_replace_re_matches_ColorResult(), colr_str_replace_re_pat_all_ColorResult(), and colr_str_replace_re_pat_ColorResult().

#### 0.6.1.2.54 ColorResult_is_ptr()

```
bool ColorResult_is_ptr (
            void * p )
```

Checks a void pointer to see if it contains a ColorResult struct.

The first member of a ColorResult is a marker.

Parameters

| | | |
|---|---|---|
| in | *p* | A void pointer to check. |

Returns

`true` if the pointer is a ColorResult, otherwise `false`.

See also

ColorResult

Referenced by _colr_free(), _colr_join(), _colr_join_array_length(), _colr_join_arrayn_size(), _colr_ptr_length(), _colr_ptr_repr(), _colr_ptr_to_str(), colr_join_arrayn(), and colr_printf_handler().

#### 0.6.1.2.55 ColorResult_length()

```
size_t ColorResult_length (
            ColorResult cres )
```

Return the length in bytes (including the null-terminator), that is needed to store the return from ColorResult_to_str() (`.result`).

Parameters

| | | |
|---|---|---|
| in | *cres* | A ColorResult to calculate the length for. |

Returns

The length of a ColorResult, possibly `0` if `.result` is `NULL`.

See also

ColorResult

Referenced by _colr_join_arrayn_size(), and _colr_ptr_length().

#### 0.6.1.2.56 ColorResult_new()

```
ColorResult ColorResult_new (
            char * s )
```

Initialize a new ColorResult with an allocated string (`char*`).

Parameters

| | | |
|---|---|---|
| `in` | *s* | An allocated string to use for the `.result` member. |

Returns

An initialized ColorResult.

See also

ColorResult

#### 0.6.1.2.57 ColorResult_repr()

```
char* ColorResult_repr (
            ColorResult cres )
```

Create a string representation for a ColorResult.

This happens to be the same as `colr_str_repr(cres.result)` right now.

Parameters

| | | |
|---|---|---|
| `in` | *cres* | A ColorResult to create the representation string for. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

ColorResult

Referenced by _colr_ptr_repr().

#### 0.6.1.2.58 ColorResult_to_ptr()

```
ColorResult* ColorResult_to_ptr (
            ColorResult cres )
```

Allocate memory for a ColorResult, fill it, and return it.

This ensure the appropriate struct marker is set, for use with Colr.

Parameters

| | | |
|---|---|---|
| in | *cres* | A ColorResult to use. |

Returns

An allocated ColorResult.
*You must `free()` the memory allocated by this function.*
If used inside of the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros, *they will `free()` the result*. Otherwise, *you are responsible for calling `free()`*.
*If allocation fails, `NULL` is returned*.

See also

ColorResult

#### 0.6.1.2.59 ColorResult_to_str()

```
char* ColorResult_to_str (
            ColorResult cres )
```

Convert a ColorResult into a string (`char*`).

This simply returns the `.result` member right now. It is used for compatibility with the colr_to_str() macro.

Parameters

| | | |
|---|---|---|
| in | *cres* | A ColorResult to use. |

Returns

A stringified-version if this ColorResult, which happens to be the `.result` member. *If you free the result of this function, the original string used to create the ColorResult will be lost.*

See also

ColorResult

Referenced by _colr_join(), _colr_ptr_to_str(), colr_join_arrayn(), colr_printf_handler(), colr_str_replace_all_ColorResult(), colr_str_replace_ColorResult(), colr_str_replace_re_all_ColorResult(), colr_str_replace_re_ColorResult(), colr_str_replace_re_match_ColorResult(), colr_str_replace_re_matches_ColorResult(), colr_str_replace_re_pat_all_ColorResult(), and colr_str_replace_re_pat_ColorResult().

#### 0.6.1.2.60 ColorText_empty()

```
ColorText ColorText_empty (
            void  )
```

Creates an "empty" ColorText with pointers set to `NULL`.

Returns

An initialized ColorText.

See also

ColorText

Referenced by ColorText_from_values(), and ColorText_set_values().

#### 0.6.1.2.61 ColorText_free()

```
void ColorText_free (
            ColorText * p )
```

Frees a ColorText and it's ColorArgs.

The text member is left alone, because it wasn't created by ColrC.

Parameters

| | | |
|---|---|---|
| in | *p* | Pointer to ColorText to free, along with it's Colr-based members. |

See also

ColorText

Referenced by _colr_free(), _colr_join(), colr_printf_handler(), colr_str_replace_all_ColorText(), colr_str_replace_ColorText(), colr_str_replace_re_all_ColorText(), colr_str_replace_re_ColorText(), colr_str_replace_re_match_ColorText(), colr_str_replace_re_matches_ColorText(), colr_str_replace_re_pat_all_ColorText(), and colr_str_replace_re_pat_ColorText().

#### 0.6.1.2.62 ColorText_free_args()

```
void ColorText_free_args (
            ColorText * p )
```

Frees the ColorArg members of a ColorText.

The ColorText itself is not free'd.

This is safe to use on a stack-allocated ColorText with heap-allocated ColorArgs.

Parameters

| | | |
|---|---|---|
| in | *p* | Pointer to a ColorText. |

See also

ColorText

Referenced by ColorText_free().

#### 0.6.1.2.63 ColorText_from_values()

```
ColorText ColorText_from_values (
            char * text,
             ...  )
```

Builds a ColorText from 1 mandatory string (`char*`), and optional fore, back, and style args (pointers to ColorArgs).

Parameters

| | | |
|---|---|---|
| in | *text* | Text to colorize (a regular string). |
| in | ... | ColorArgs for fore, back, and style, in any order. |

Returns

An initialized ColorText struct.

See also

ColorText

#### 0.6.1.2.64 ColorText_has_arg()

```
bool ColorText_has_arg (
            ColorText ctext,
            ColorArg carg )
```

Checks to see if a ColorText has a certain ColorArg value set.

Uses ColorArg_eq() to inspect the `fore`, `back`, and `style` members.

Parameters

| | | |
|---|---|---|
| `in` | *ctext* | The ColorText to inspect. |
| `in` | *carg* | The ColorArg to look for. |

Returns

`true` if the `fore`, `back`, or `style` arg matches `carg`, otherwise `false`.

See also

ColorText

#### 0.6.1.2.65 ColorText_has_args()

```
bool ColorText_has_args (
            ColorText ctext )
```

Checks to see if a ColorText has any argument values set.

Parameters

| | | |
|---|---|---|
| `in` | *ctext* | A ColorText to check. |

Returns

`true` if `.fore`, `.back`, or `.style` is set to a non-empty ColorArg, otherwise `false`.

See also

ColorText

#### 0.6.1.2.66 ColorText_is_empty()

```
bool ColorText_is_empty (
            ColorText ctext )
```

Checks to see if a ColorText has no usable values.

A ColorText is considered "empty" if the `.text`, `.fore`, `.back`, and `.style` pointers are `NULL`, and the `.just` member is set to an "empty" ColorJustify.

Parameters

| | | |
|---|---|---|
| `in` | *ctext* | The ColorText to check. |

Returns

`true` if the ColorText is empty, otherwise `false`.

See also

ColorText
ColorText_empty

#### 0.6.1.2.67 ColorText_is_ptr()

```
bool ColorText_is_ptr (
            void * p )
```

Checks a void pointer to see if it contains a ColorText struct.

The first member of a ColorText is a marker.

Parameters

| | | |
|---|---|---|
| `in` | *p* | A void pointer to check. |

Returns

`true` if the pointer is a ColorText, otherwise `false`.

See also

ColorText

Referenced by _colr_free(), _colr_join(), _colr_join_array_length(), _colr_join_arrayn_size(), _colr_↩ ptr_length(), _colr_ptr_repr(), _colr_ptr_to_str(), colr_join_arrayn(), and colr_printf_handler().

#### 0.6.1.2.68 ColorText_length()

```
size_t ColorText_length (
            ColorText ctext )
```

Returns the length in bytes needed to allocate a string (`char*`) built with ColorText_to_str() with the current `text`, `fore`, `back`, and `style` members.

Parameters

| | | |
|---|---|---|
| `in` | *ctext* | ColorText to use. |

Returns

The length (`size_t`) needed to allocate a ColorText's string, or `1` (size of an empty string) for invalid/empty arg types/values.

See also

ColorText

Referenced by _colr_join_arrayn_size(), _colr_ptr_length(), and ColorText_to_str().

#### 0.6.1.2.69 ColorText_repr()

```
char* ColorText_repr (
            ColorText ctext )
```

Allocate a string (`char*`) representation for a ColorText.

Parameters

| | | |
|---|---|---|
| `in` | *ctext* | ColorText to get the string representation for. |

Returns

Allocated string for the ColorText.

See also

ColorText

Referenced by _colr_ptr_repr().

#### 0.6.1.2.70 ColorText_set_just()

```
ColorText* ColorText_set_just (
            ColorText * ctext,
            ColorJustify cjust )
```

Set the ColorJustify method for a ColorText, and return the ColorText.

This is to facilitate the justification macros. If you already have a pointer to a ColorText, you can just do `ctext->just = just;`. The purpose of this is to allow `ColorText_set_just(ColorText_to_ptr(...), ...)` to work.

Parameters

| | | |
|---|---|---|
| out | *ctext* | The ColorText to set the justification method for. |
| in | *cjust* | The ColorJustify struct to use. |

Returns

The same pointer that was given as `ctext`.

See also

ColorText

#### 0.6.1.2.71 ColorText_set_values()

```
void ColorText_set_values (
            ColorText * ctext,
            char * text,
             ...  )
```

Initializes an existing ColorText from 1 mandatory string (`char*`), and optional fore, back, and style args (pointers to ColorArgs).

Parameters

| | | |
|---|---|---|
| out | *ctext* | A ColorText to initialize with values. |
| in | *text* | Text to colorize (a regular string). |
| in | ... | A `va_list` with ColorArgs pointers for fore, back, and style, in any order. |

Returns

An initialized ColorText struct.

See also

ColorText

#### 0.6.1.2.72 ColorText_to_ptr()

```
ColorText* ColorText_to_ptr (
            ColorText ctext )
```

Copies a ColorText into allocated memory and returns the pointer.

You must free() the memory if you call this directly.

Parameters

| | | |
|---|---|---|
| in | *ctext* | ColorText to copy/allocate for. |

Returns

Pointer to a heap-allocated ColorText.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

ColorText

#### 0.6.1.2.73 ColorText_to_str()

```
char* ColorText_to_str (
            ColorText ctext )
```

Stringifies a ColorText struct, creating a mix of escape codes and text.

Parameters

| | | |
|---|---|---|
| in | *ctext* | ColorText to stringify. |

Returns

An allocated string with text/escape-codes.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*. If the ColorText has a `NULL` `.text` member, `NULL` is returned.

See also

ColorText

Referenced by _colr_join(), _colr_ptr_to_str(), colr_join_arrayn(), colr_printf_handler(), colr_str_replace_all_ColorText(), colr_str_replace_ColorText(), colr_str_replace_re_all_ColorText(), colr_str_replace_re_ColorText(), colr_str_replace_re_match_ColorText(), colr_str_replace_re_matches_ColorText(), colr_str_replace_re_pat_all_ColorText(), and colr_str_replace_re_pat_ColorText().

#### 0.6.1.2.74 ColorType_eq()

```
bool ColorType_eq (
            ColorType a,
            ColorType b )
```

Compares two ColorTypes.

This is used to implement colr_eq().

Parameters

| | | |
|---|---|---|
| `in` | *a* | The first ColorType to compare. |
| `in` | *b* | The second ColorType to compare. |

Returns

`true` if they are equal, otherwise `false`.

See also

ColorType

#### 0.6.1.2.75 ColorType_from_str()

```
ColorType ColorType_from_str (
            const char * arg )
```

Determine which type of color value is desired by name.

Example:

- "red" == TYPE_BASIC
- "253" == TYPE_EXTENDED
- "123,55,67" == TYPE_RGB

Parameters

| | | |
|---|---|---|
| in | *arg* | Color name to get the ColorType for. |

Return values

| | |
|---|---|
| *ColorType* | value on success. |
| *TYPE_INVALID* | for invalid color names/strings. |
| *TYPE_INVALID_EXT_RANGE* | for ExtendedValues outside of 0-255. |
| *TYPE_INVALID_RGB_RANGE* | for rgb values outside of 0-255. |

See also

ColorType

#### 0.6.1.2.76 ColorType_is_invalid()

```
bool ColorType_is_invalid (
            ColorType type )
```

Check to see if a ColorType value is considered invalid.

Parameters

| | | |
|---|---|---|
| in | *type* | ColorType value to check. |

Returns

`true` if the value is considered invalid, otherwise `false`.

See also

ColorType

#### 0.6.1.2.77 ColorType_is_valid()

```
bool ColorType_is_valid (
            ColorType type )
```

Check to see if a ColorType value is considered valid.

Parameters

| | | |
|---|---|---|
| in | *type* | ColorType value to check. |

Returns

`true` if the value is considered valid, otherwise `false`.

See also

ColorType

#### 0.6.1.2.78 ColorType_repr()

```
char* ColorType_repr (
            ColorType type )
```

Creates a string (`char*`) representation of a ColorType.

Parameters

| | | |
|---|---|---|
| in | *type* | A ColorType to get the type from. |

Returns

A pointer to an allocated string.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

ColorType

#### 0.6.1.2.79 ColorType_to_str()

```
char* ColorType_to_str (
            ColorType type )
```

Create a human-friendly string (`char*`) representation for a ColorType.

Parameters

| | | |
|---|---|---|
| in | *type* | A ColorType to get the name for. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

ColorType

Referenced by ColorValue_example().

#### 0.6.1.2.80 ColorValue_empty()

```
ColorValue ColorValue_empty (
            void  )
```

Create an "empty" ColorValue.

This is used with ColorArg_empty() to build ColorArgs that don't do anything, where using `NULL` has a different meaning inside the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros.

Returns

`(ColorValue){.type=TYPE_NONE, .basic=0, .ext=0, .rgb=(RGB){0, 0, 0}}`

See also

ColorArg
ColorArg_empty
ColorArg_is_empty
ColorValue_is_empty

#### 0.6.1.2.81 ColorValue_eq()

```
bool ColorValue_eq (
            ColorValue a,
            ColorValue b )
```

Compares two ColorValue structs.

They are considered "equal" if all of their members match.

Parameters

| | | |
|---|---|---|
| in | *a* | First ColorValue to compare. |
| in | *b* | Second ColorValue to compare. |

Returns

`true` if they are equal, otherwise `false`.

See also

ColorValue

Referenced by ColorArg_eq().

#### 0.6.1.2.82 ColorValue_example()

```
char* ColorValue_example (
            ColorValue cval )
```

Create a string (`char*`) representation of a ColorValue with a human-friendly type/name.

Parameters

| | | |
|---|---|---|
| `in` | *cval* | A ColorValue to get an example string for. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

ColorValue

Referenced by ColorArg_example().

#### 0.6.1.2.83 ColorValue_from_esc()

```
ColorValue ColorValue_from_esc (
            const char * s )
```

Convert an escape-code string (`char*`) into a ColorValue.

Parameters

| | | |
|---|---|---|
| `in` | *s* | An escape-code string to parse.<br>*Must be `null`-terminated.* |

Returns

A ColorValue (with no fore/back information, only the color type and value).

Return values

| | |
|---|---|
| *For* | invalid strings, the `.type` member can be one of:<br>• TYPE_INVALID<br>• TYPE_INVALID_EXT_RANGE<br>• TYPE_INVALID_RGB_RANGE |

See also

ColorValue
ColorArg_from_esc

Referenced by ColorArg_from_esc().

#### 0.6.1.2.84 ColorValue_from_str()

```
ColorValue ColorValue_from_str (
            const char * s )
```

Create a ColorValue from a known color name, or RGB string (`char*`).

Parameters

| | | |
|---|---|---|
| `in` | *s* | A string to parse the color name from (can be an RGB string). |

Returns

A ColorValue (with no fore/back information, only the color type and value).

Return values

| | |
|---|---|
| *For* | invalid strings, the `.type` member can be one of:<br>• TYPE_INVALID<br>• TYPE_INVALID_EXT_RANGE<br>• TYPE_INVALID_RGB_RANGE |

See also

ColorValue

Referenced by ColorArg_from_str().

#### 0.6.1.2.85 ColorValue_from_value()

```
ColorValue ColorValue_from_value (
            ColorType type,
            void * p )
```

Used with the color_val macro to dynamically create a ColorValue based on it's argument type.

Parameters

| | | |
|---|---|---|
| in | *type* | A ColorType value, to mark the type of ColorValue. |
| in | *p* | A pointer to either a BasicValue, ExtendedValue, or a RGB. |

Returns

A ColorValue struct with the appropriate `.type` member set for the value that was passed. For invalid types the `.type` member may be set to one of:

- TYPE_INVALID
- TYPE_INVALID_EXT_RANGE
- TYPE_INVALID_RGB_RANGE

See also

ColorValue

Referenced by ColorArg_from_BasicValue(), ColorArg_from_ExtendedValue(), ColorArg_from_RGB(), ColorArg_from_StyleValue(), ColorValue_from_esc(), and ColorValue_from_str().

#### 0.6.1.2.86 ColorValue_has_BasicValue()

```
bool ColorValue_has_BasicValue (
            ColorValue cval,
            BasicValue bval )
```

Checks to see if a ColorValue has a BasicValue set.

Parameters

| | | |
|---|---|---|
| in | *cval* | ColorValue to check. |
| in | *bval* | BasicValue to look for. |

Returns

`true` if the ColorValue has the exact BasicValue set.

See also

ColorValue

#### 0.6.1.2.87 ColorValue_has_ExtendedValue()

```
bool ColorValue_has_ExtendedValue (
            ColorValue cval,
            ExtendedValue eval )
```

Checks to see if a ColorValue has a ExtendedValue set.

Parameters

| | | |
|---|---|---|
| in | *cval* | ColorValue to check. |
| in | *eval* | ExtendedValue to look for. |

Returns

`true` if the ColorValue has the exact ExtendedValue set.

See also

ColorValue

#### 0.6.1.2.88 ColorValue_has_RGB()

```
bool ColorValue_has_RGB (
            ColorValue cval,
            RGB rgb )
```

Checks to see if a ColorValue has a RGB value set.

Parameters

| | | |
|---|---|---|
| in | *cval* | ColorValue to check. |
| in | *rgb* | RGB value to look for. |

Returns

`true` if the ColorValue has the exact RGB value set.

See also

ColorValue

#### 0.6.1.2.89 ColorValue_has_StyleValue()

```
bool ColorValue_has_StyleValue (
            ColorValue cval,
            StyleValue sval )
```

Checks to see if a ColorValue has a StyleValue set.

Parameters

| | | |
|---|---|---|
| in | *cval* | ColorValue to check. |
| in | *sval* | StyleValue to look for. |

Returns

`true` if the ColorValue has the exact StyleValue set.

See also

ColorValue

#### 0.6.1.2.90 ColorValue_is_empty()

```
bool ColorValue_is_empty (
            ColorValue cval )
```

Checks to see if a ColorValue is an empty placeholder.

Parameters

| | | |
|---|---|---|
| in | *cval* | ColorValue to check. |

Returns

`true` if the ColorValue is "empty", otherwise `false`.

See also

ColorValue
ColorValue_empty
ColorArg_empty
ColorArg_is_empty

#### 0.6.1.2.91 ColorValue_is_invalid()

```
bool ColorValue_is_invalid (
            ColorValue cval )
```

Checks to see if a ColorValue holds an invalid value.

Parameters

| | | |
|---|---|---|
| in | *cval* | ColorValue struct to check. |

Returns

`true` if the value is invalid, otherwise `false`.

See also

ColorValue

Referenced by ColorArg_from_esc().

#### 0.6.1.2.92 ColorValue_is_valid()

```
bool ColorValue_is_valid (
            ColorValue cval )
```

Checks to see if a ColorValue holds a valid value.

Parameters

| | | |
|---|---|---|
| in | *cval* | ColorValue struct to check. |

Returns

`true` if the value is valid, otherwise `false`.

See also

ColorValue

#### 0.6.1.2.93 ColorValue_length()

```
size_t ColorValue_length (
            ArgType type,
            ColorValue cval )
```

Returns the length in bytes needed to allocate a string (`char*`) built with ColorValue_to_esc() with the specified ArgType and ColorValue.

Parameters

| | | |
|---|---|---|
| in | *type* | ArgType (FORE, BACK, STYLE) |
| in | *cval* | ColorValue to use. |

Returns

The length (size_t) needed to allocate a ColorValue's string, or 1 (size of an empty string) for invalid/empty arg types/values.

See also

ColorValue

Referenced by ColorArg_length().

#### 0.6.1.2.94 ColorValue_repr()

```
char* ColorValue_repr (
            ColorValue cval )
```

Creates a string (char*) representation of a ColorValue.

Parameters

| | | |
|---|---|---|
| in | *cval* | A ColorValue to get the type and value from. |

Returns

A pointer to an allocated string.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned.*

See also

ColorValue

Referenced by ColorArg_repr().

#### 0.6.1.2.95 ColorValue_to_esc()

```
char* ColorValue_to_esc (
            ArgType type,
            ColorValue cval )
```

Converts a ColorValue into an escape code string (char*).

Parameters

| | | |
|---|---|---|
| in | *type* | ArgType (FORE, BACK, STYLE) to build the escape code for. |
| in | *cval* | ColorValue to get the color value from. |

Returns

An allocated string with the appropriate escape code. For invalid values, an empty string is returned.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

ColorValue

Referenced by ColorArg_to_esc().

#### 0.6.1.2.96 ColorValue_to_esc_s()

```
bool ColorValue_to_esc_s (
            char * dest,
            ArgType type,
            ColorValue cval )
```

Converts a ColorValue into an escape code string (`char*`) and fills the destination string.

For invalid ArgType/ColorValue combinations, `dest[0]` is set to "\0".

Parameters

| | | |
|---|---|---|
| out | *dest* | Destination string for the escape code string. *Must have room for the code type being used*. |
| in | *type* | ArgType (FORE, BACK, STYLE) to build the escape code for. |
| in | *cval* | ColorValue to get the color value from. |

Returns

`true` if a proper ArgType/ColorValue combination was used, otherwise `false`.

See also

ColorValue

Referenced by ColorArg_to_esc_s().

#### 0.6.1.2.97 colr_alloc_regmatch()

```
regmatch_t* colr_alloc_regmatch (
            regmatch_t match )
```

Allocates space for a regmatch_t, initializes it, and returns a pointer to it.

Parameters

| | | |
|---|---|---|
| in | *match* | A `regmatch_t` to allocate for and copy. |

Returns

An allocated copy of the `regmatch_t`.

Referenced by colr_re_matches().

#### 0.6.1.2.98 colr_append_reset()

```
void colr_append_reset (
            char * s )
```

Appends CODE_RESET_ALL to a string (`char*`), but makes sure to do it before any newlines.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to append to. *Must have extra room for CODE_RESET_ALL*. *Must be `null`-terminated*. |

Referenced by _colr_join(), _rainbow(), ColorText_to_str(), and colr_join_arrayn().

#### 0.6.1.2.99 colr_char_escape_char()

```
char colr_char_escape_char (
            const char c )
```

Returns the char needed to represent an escape sequence in C.

The following characters are supported:

| Escape Sequence | Description Representation |
|---|---|
| \' | single quote |
| \" | double quote |
| \? | question mark |
| \\ | backslash |
| \a | audible bell |

| Escape Sequence | Description Representation |
|---:|---|
| \ b | backspace |
| \ f | form feed - new page |
| \ n | line feed - new line |
| \ r | carriage return |
| \ t | horizontal tab |
| \ v | vertical tab |

Parameters

| | | |
|---|---|---|
| in | *c* | The character to check. |

Returns

The letter, without a backslash, needed to create an escape sequence. If the char doesn't need an escape sequence, it is simply returned.

Referenced by colr_str_repr().

#### 0.6.1.2.100 colr_char_in_str()

```
bool colr_char_in_str (
            const char * s,
            const char c )
```

Determines if a character exists in the given string (`char*`).

Parameters

| | | |
|---|---|---|
| in | *c* | Character to search for. |
| in | *s* | String to check.<br>Input *must be `null`-terminated*. |

Returns

`true` if `c` is found in `s`, otherwise `false`.

Referenced by colr_str_chars_lcount(), and colr_str_lstrip_chars().

#### 0.6.1.2.101 colr_char_is_code_end()

```
bool colr_char_is_code_end (
            const char c )
```

Determines if a character is suitable for an escape code ending.

`m` is used as the last character in color codes, but other characters can be used for escape sequences (such as "\x1b[2A", cursor up). Actual escape code endings can be in the range (`char`) 64-126 (inclusive).

Since ColrC only deals with color codes and maybe some cursor/erase codes, this function tests if the character is either `A-Z` or `a-z`.

For more information, see: https://en.wikipedia.org/wiki/ANSI_escape_code

Parameters

| | | |
|---|---|---|
| in | *c* | Character to test. |

Returns

`true` if the character is a possible escape code ending, otherwise `false`.

Referenced by colr_str_code_count(), colr_str_code_len(), colr_str_get_codes(), colr_str_is_codes(), colr_str_noncode_len(), and colr_str_strip_codes().

#### 0.6.1.2.102 colr_char_repr()

```
char* colr_char_repr (
            char c )
```

Creates a string (`char*`) representation for a char.

Parameters

| | | |
|---|---|---|
| in | *c* | Value to create the representation for. |

Returns

An allocated string.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

Referenced by ColorJustify_repr().

#### 0.6.1.2.103 colr_char_should_escape()

```
bool colr_char_should_escape (
            const char c )
```

Determines if an ascii character has an escape sequence in C.

The following characters are supported:

| Escape Sequence | Description Representation |
|---|---|
| \' | single quote |
| \" | double quote |
| \? | question mark |
| \\ | backslash |
| \a | audible bell |
| \b | backspace |
| \f | form feed - new page |
| \n | line feed - new line |
| \r | carriage return |
| \t | horizontal tab |
| \v | vertical tab |

Parameters

| | | |
|---|---|---|
| in | *c* | The character to check. |

Returns

`true` if the character needs an escape sequence, otherwise `false`.

Referenced by colr_str_repr().

#### 0.6.1.2.104 colr_check_marker()

```
bool colr_check_marker (
            uint32_t marker,
            void * p )
```

Checks an unsigned int against the individual bytes behind a pointer's value.

This helps to guard against overflows, because only a single byte is checked at a time. If any byte doesn't match the marker, `false` is immediately returned, instead of continuing past the pointer's bounds.

Parameters

| | | |
|---|---|---|
| in | *marker* | A colr marker, like COLORARG_MARKER, COLORTEXT_MARKER, etc. |
| in | *p* | A pointer to check, to see if it starts with the marker. |

Returns

`true` if all bytes match the marker, otherwise `false`.

See also

ColorArg_is_ptr
ColorText_is_ptr

Referenced by _colr_is_last_arg(), ColorArg_is_ptr(), ColorResult_is_ptr(), and ColorText_is_ptr().

#### 0.6.1.2.105 colr_empty_str()

```
char* colr_empty_str (
            void  )
```

Allocates an empty string (`char*`).

This is for keeping the interface simple, so the return values from color functions with invalid values can be consistent.

Returns

Pointer to an allocated string consisting of '\0'.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

Referenced by colr_str_center(), colr_str_ljust(), colr_str_replace_re_match(), colr_str_rjust(), and colr_str_strip_codes().

#### 0.6.1.2.106 colr_free_re_matches()

```
void colr_free_re_matches (
            regmatch_t ** matches )
```

Free an array of allocated `regmatch_t`, like the return from colr_re_matches().

Parameters

| | | |
|---|---|---|
| `out` | *matches* | A pointer to an array of `regmatch_t` pointers. |

Referenced by colr_str_replace_re_pat_all().

#### 0.6.1.2.107 colr_join_array()

```
char* colr_join_array (
            void * joinerp,
            void * ps )
```

Join an array of strings (`char*`), `ColorArg`s, or `ColorText`s by another string (`char*`), ColorArg, or ColorText.

Parameters

| | | |
|---|---|---|
| in | *joinerp* | The joiner (any ColorArg, ColorText, or string (char*)). |
| in | *ps* | An array of pointers to ColorArgs, ColorTexts, or strings (char*). The array must have NULL as the last item. |

Returns

An allocated string with the result.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr
colr_join
colr_join_arrayn

Examples:

colr_join_example.c.

### 0.6.1.2.108 colr_join_arrayn()

```
char* colr_join_arrayn (
            void * joinerp,
            void * ps,
            size_t count )
```

Join an array of strings (char*), ColorArgs, or ColorTexts by another string (char*), ColorArg, or ColorText.

Parameters

| | | |
|---|---|---|
| in | *joinerp* | The joiner (any ColorArg, ColorText, or string (char*)). |
| in | *ps* | An array of pointers to ColorArgs, ColorTexts, or strings (char*). The array must have at least a length of count, unless a NULL element is placed at the end. |
| in | *count* | The total number of items in the array. |

Returns

An allocated string with the result.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*. If any parameter is NULL, NULL is returned.

See also

colr
colr_join

Referenced by colr_join_array().

#### 0.6.1.2.109 colr_mb_len()

```
size_t colr_mb_len (
            const char * s,
            size_t length )
```

Like `mbrlen`, except it will return the length of the next N (`length`) multibyte characters in bytes.

/details Unlike colr_str_mb_len(), which returns the number of multibyte characters, this function will return the number of bytes that make up the next number (`length`) of multibyte characters.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to check. |
| `in` | *length* | Number of multibyte characters to get the length for. |

Returns

The number of bytes parsed in `s` to get at least `length` multibyte characters.

Return values

| | |
|---|---|
| *0* | if `s` is `NULL`/empty, or `length` is `0`. |
| *(size_t)-1* | if an invalid multibyte sequence is found at the start of `s`. |

See also

colr_str_mb_len
colr_is_valid_mblen

Referenced by _rainbow().

#### 0.6.1.2.110 colr_printf_handler()

```
int colr_printf_handler (
            FILE * fp,
            const struct printf_info * info,
            const void *const * args )
```

Handles printing with printf for Colr objects.

This function matches the required `typedef` in `printf.h` (`printf_function`), for handling a custom printf format char with `register_printf_specifier`.

Attention

This feature uses a GNU extension, and is only available when COLR_GNU is defined. See the documentation for COLR_GNU.

Parameters

| | | |
|---|---|---|
| in | *fp* | FILE pointer for output. |
| in | *info* | Info from printf about how to format the argument. |
| in | *args* | Argument list (with only 1 argument), containing a ColorArg, ColorResult, ColorText, or string (char*) to format. |

Returns

The number of characters written.

Referenced by colr_printf_register().

#### 0.6.1.2.111 colr_printf_info()

```
int colr_printf_info (
            const struct printf_info * info,
            size_t n,
            int * argtypes,
            int * sz )
```

Handles the arg count/size for the Colr printf handler.

This function matches the required `typedef` in `printf.h` (`printf_arginfo_size_function`) for handling a custom printf format char with `register_printf_specifier`.

Attention

This feature uses a GNU extension, and is only available when COLR_GNU is defined. See the documentation for COLR_GNU.

Parameters

| | | |
|---|---|---|
| in | *info* | Info from printf about how to format the argument. |
| in | *n* | Number of arguments for the format char. |
| out | *argtypes* | Type of arguments being handled, from an `enum` defined in `printf`. Colr uses/sets one argument, a `PA_POINTER` type. |
| out | *sz* | Size of the arguments. Not used in Colr. |

Returns

The number of argument types set in `argtypes`.

Referenced by colr_printf_register().

#### 0.6.1.2.112 colr_printf_register()

```
void colr_printf_register (
            void  )
```

Registers COLR_FMT_CHAR to handle Colr objects in the printf-family functions.

This function only needs to be called once and `register_printf_specifier` is only called the first time this function is called.

Attention

This feature uses a GNU extension, and is only available when COLR_GNU is defined. See the documentation for COLR_GNU.

#### 0.6.1.2.113 colr_re_matches()

```
regmatch_t** colr_re_matches (
            const char * s,
            regex_t * repattern )
```

Returns all `regmatch_t` matches for regex pattern in a string (`char*`).

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to search. |
| `in` | *repattern* | The pattern to look for. |

Returns

A pointer to an allocated array of `regmatch_t*`, or `NULL` if `s` is `NULL` or `repattern` is `NULL`. The last member is always `NULL`.
*You must `free()` the memory allocated by this function.*

Examples:

colr_replace_all_example.c.

Referenced by colr_str_replace_re_pat_all().

#### 0.6.1.2.114 colr_set_locale()

```
bool colr_set_locale (
            void  )
```

Sets the locale to `(LC_ALL, "")` if it hasn't already been set.

This is used for functions dealing with multibyte strings.

Returns

`true` if the locale had to be set, `false` if it was already set.

Referenced by colr_mb_len(), and colr_str_mb_len().

#### 0.6.1.2.115 colr_str_array_contains()

```
bool colr_str_array_contains (
            char ** lst,
            const char * s )
```

Determine if a string (`char*`) is in an array of strings (`char**`, where the last element is `NULL`).

Parameters

| | | |
|---|---|---|
| `in` | *lst* | The string array to look in. |
| `in` | *s* | The string to look for. |

Returns

`true` if the string is found, otherwise `false`.

Return values

| | |
|---|---|
| *<tt>false</tt>* | if `lst` is `NULL` or `s` is `NULL`. |

Referenced by colr_str_get_codes().

#### 0.6.1.2.116 colr_str_array_free()

```
void colr_str_array_free (
            char ** ps )
```

Free an allocated array of strings, including the array itself.

Each individual string will be released, and finally the allocated memory for the array of pointers will be released.

Parameters

| | | |
|---|---|---|
| in | *ps* | A pointer to an array of strings. |

Referenced by ColorArgs_from_str().

#### 0.6.1.2.117 colr_str_center()

```
char* colr_str_center (
            const char * s,
            int width,
            const char padchar )
```

Center-justifies a string (`char*`), ignoring escape codes when measuring the width.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to justify.<br>Input *must be `null`-terminated*. |
| in | *width* | The overall width for the resulting string. If set to '0', the terminal width will be used from colr_term_size(). |
| in | *padchar* | The character to pad with. If '0', then '' is used. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_str_ljust
colr_str_rjust
colr_term_size

Referenced by colr_printf_handler().

#### 0.6.1.2.118 colr_str_char_count()

```
size_t colr_str_char_count (
            const char * s,
            const char c )
```

Counts the number of characters (`c`) that are found in a string (`char*`) (`s`).

Returns `0` if `s` is `NULL`, or `c` is ''\0''.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to examine.<br>*Must be `null`-terminated*. |
| in | *c* | The character to count.<br>*Must not be `0`*. |

Returns

The number of times `c` occurs in `s`.

Referenced by _rainbow().

#### 0.6.1.2.119 colr_str_char_lcount()

```
size_t colr_str_char_lcount (
            const char * s,
            const char c )
```

Counts the number of characters (`c`) that are found at the beginning of a string (`char*`) (`s`).

Returns `0` if `s` is `NULL`, `c` is "\0", or the string doesn't start with `c`.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to examine.<br>*Must be `null`-terminated*. |
| in | *c* | The character to count.<br>*Must not be `0`*. |

Returns

The number of times `c` occurs at the start of `s`.

Referenced by colr_str_lstrip_char().

#### 0.6.1.2.120 colr_str_chars_lcount()

```
size_t colr_str_chars_lcount (
            const char *restrict s,
            const char *restrict chars )
```

Counts the number of characters that are found at the beginning of a string (`char*`) (`s`), where the character can be any of `chars`.

Returns `0` if `s` is `NULL`/empty, `chars` is `NULL`/empty, or the string doesn't start with any of the characters in `chars`.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to examine.<br>*Must be `null`-terminated*. |
| in | *chars* | The characters to count, in any order.<br>*Must not be `0`*. |

Returns

The number of times a character in `chars` occurs at the start of `s`.

Referenced by colr_str_lstrip_chars().

#### 0.6.1.2.121 colr_str_code_count()

```
size_t colr_str_code_count (
            const char * s )
```

Return the number of escape-codes in a string (`char*`).

Parameters

| | | |
|---|---|---|
| in | *s* | A string to count the escape-codes for.<br>*Must be `null`-terminated*. |

Returns

The number of escape-codes, or `0` if `s` is `NULL`, or doesn't contain any escape-codes.

Referenced by colr_str_get_codes().

#### 0.6.1.2.122 colr_str_code_len()

```
size_t colr_str_code_len (
            const char * s )
```

Return the number of bytes that make up all the escape-codes in a string (`char*`).

Parameters

| | | |
|---|---|---|
| in | *s* | A string to count the code-chars for.<br>*Must be `null`-terminated*. |

Returns

The number of escape-code characters, or `0` if `s` is `NULL`, or doesn't contain any escape-codes.

#### 0.6.1.2.123 colr_str_copy()

```
char* colr_str_copy (
            char *restrict dest,
            const char *restrict src,
            size_t length )
```

Copies a string (`char*`) like strncpy, but ensures null-termination.

If `src` is `NULL`, or `dest` is `NULL`, `NULL` is returned.

If `src` does not contain a null-terminator, *this function will truncate at `length` characters*.

If `src` is an empty string, then `dest[0]` will be "\0" (an empty string).

A null-terminator is always appended to `dest`.

`src` and `dest` must not overlap.

Parameters

| | | |
|---|---|---|
| `in` | *dest* | Memory allocated for new string. *Must have room for `strlen(src) + 1` or `length + 1`.* |
| `in` | *src* | Source string to copy. |
| `in` | *length* | Maximum characters to copy. *This does not include the null-terminator.* Usually set to `strlen(dest)`. |

Returns

On success, a pointer to dest is returned.

#### 0.6.1.2.124 colr_str_ends_with()

```
bool colr_str_ends_with (
            const char *restrict s,
            const char *restrict suffix )
```

Determine if one string (`char*`) ends with another.

`str` and `suffix` must not overlap.

Parameters

| | | |
|---|---|---|
| in | *s* | String to check.<br>*Must be null-terminated.* |
| in | *suffix* | Suffix to check for.<br>*Must be null-terminated.* |

Returns

True if str ends with suffix.
False if either is NULL, or the string doesn't end with the suffix.

Referenced by colr_append_reset().

#### 0.6.1.2.125 colr_str_get_codes()

```
char** colr_str_get_codes (
            const char * s,
            bool unique )
```

Get an array of escape-codes from a string (char*).

This function copies the escape-code strings, and the pointers to the heap, if any escape-codes are found in the string.

colr_str_array_free() can be used to easily free() the result of this function.

Parameters

| | | |
|---|---|---|
| in | *s* | A string to get the escape-codes from.<br>*Must be null-terminated.* |
| in | *unique* | Whether to only include *unique* escape codes. |

Returns

An allocated array of string (char*) pointers, where the last element is NULL.
*You must free() the memory allocated by this function.*

Return values

| | |
|---|---|
| *If* | s is NULL, or empty, or there are otherwise no escape-codes found in the string, or allocation fails for the strings/array, then NULL is returned. |
| *On* | success, there will be at least two pointers behind the return value. The last pointer is always NULL. |

Referenced by ColorArgs_from_str().

#### 0.6.1.2.126 colr_str_has_codes()

```
bool colr_str_has_codes (
            const char * s )
```

Determines if a string (`char*`) has ANSI escape codes in it.

This will detect any ansi escape code, not just colors.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to check. Can be `NULL`.<br>Input *must be `null`-terminated*. |

Returns

`true` if the string has at least one escape code, otherwise `false`.

See also

colr_str_is_codes

#### 0.6.1.2.127 colr_str_hash()

```
ColrHash colr_str_hash (
            const char * s )
```

Hash a string using djb2.

This is only used for simple, short, string (`char*`) hashing. It is not designed for cryptography.

There are some notes about collision rates for this function here.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to hash.<br>*Must be `null`-terminated*. |

Returns

A ColrHash (`unsigned long`) value with the hash.

Return values

| | |
|---|---|
| *0* | if s is `NULL`. |
| *COLR_HASH_SEED* | if s is an empty string. |

Referenced by colr_str_array_contains().

#### 0.6.1.2.128 colr_str_is_all()

```
bool colr_str_is_all (
            const char * s,
            const char c )
```

Determines whether a string (`char*`) consists of only one character, possibly repeated.

Parameters

| | | |
|---|---|---|
| in | *s* | String to check. |
| in | *c* | Character to test for. Must not be `0`. |

Returns

`true` if `s` contains only the character `c`, otherwise `false`.

#### 0.6.1.2.129 colr_str_is_codes()

```
bool colr_str_is_codes (
            const char * s )
```

Determines if a string (`char*`) is composed entirely of escape codes.

Returns `false` if the string is `NULL`, or empty.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to check.<br>Input *must be `null`-terminated*. |

Returns

`true` if the string is escape-codes only, otherwise `false`.

See also

colr_str_has_codes

#### 0.6.1.2.130 colr_str_is_digits()

```
bool colr_str_is_digits (
            const char * s )
```

Determines whether all characters in a string (`char*`) are digits.

If `s` is NULL or an empty string (""), `false` is returned.

Parameters

| | | |
|---|---|---|
| `in` | *s* | String to check.<br>Input *must be `null`-terminated*. |

Returns

`true` if all characters are digits (0-9), otherwise `false`.

Referenced by ExtendedValue_from_str().

#### 0.6.1.2.131 colr_str_ljust()

```
char* colr_str_ljust (
            const char * s,
            int width,
            const char padchar )
```

Left-justifies a string (`char*`), ignoring escape codes when measuring the width.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to justify.<br>Input *must be `null`-terminated*. |
| `in` | *width* | The overall width for the resulting string. If set to '0', the terminal width will be used from colr_term_size(). |
| `in` | *padchar* | The character to pad with. If '0', then ' ' is used. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_str_center
colr_str_rjust
colr_term_size

Referenced by colr_printf_handler().

#### 0.6.1.2.132 colr_str_lower()

```
void colr_str_lower (
            char * s )
```

Converts a string (`char*`) into lower case in place.

Input *must be `null`-terminated*.

If `s` is `NULL`, nothing is done.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The input string to convert to lower case. |

#### 0.6.1.2.133 colr_str_lstrip()

```
size_t colr_str_lstrip (
            char *restrict dest,
            const char *restrict s,
            size_t length,
            const char c )
```

Strip a leading character from a string (`char*`), filling another string (`char*`) with the result.

`dest` and `s` should not overlap.

Parameters

| | | |
|---|---|---|
| `out` | *dest* | Destination `char` array. Must have room for `strlen(s) + 1`. |
| `in` | *s* | String to strip the character from. |
| `in` | *length* | Length of `s`, the input string. |
| `in` | *c* | Character to strip. If set to `0`, all whitespace characters will be used (' ', '\n', '\t', '\v', '\f', '\r'). |

Returns

The number of `c` characters removed. May return `0` if `s` is `NULL`/empty, `dest` is `NULL`.

Referenced by colr_str_lstrip_char(), and RGB_from_hex().

#### 0.6.1.2.134 colr_str_lstrip_char()

```
char* colr_str_lstrip_char (
            const char * s,
            const char c )
```

Strips a leading character from a string (`char*`), and allocates a new string with the result.

Parameters

| | | |
|---|---|---|
| in | *s* | String to strip the character from. |
| in | *c* | Character to strip. If set to `0`, all whitespace characters will be used (' ', '\n', '\t'). |

Returns

An allocated string with the result. May return `NULL` if `s` is `NULL`/empty.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

#### 0.6.1.2.135 colr_str_lstrip_chars()

```
char* colr_str_lstrip_chars (
            const char *restrict s,
            const char *restrict chars )
```

Removes certain characters from the start of a string (`char*`) and allocates a new string with the result.

The order of the characters in `chars` does not matter. If any of them are found at the start of a string, they will be removed.

```
colr_str_lstrip_chars("aabbccTEST", "bca") == "TEST"
```

`s` and `chars` must not overlap.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to strip. `s` *Must be `null`-terminated*. |
| in | *chars* | A string of characters to remove. Each will be removed from the start of the string. `chars` *Must be `null`-terminated*. |

Returns

An allocated string with the result. May return NULL if `s` or `chars` is NULL.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

#### 0.6.1.2.136 colr_str_mb_len()

```
size_t colr_str_mb_len (
            const char * s )
```

Returns the number of characters in a string (`char*`), taking into account possibly multibyte characters.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to get the length of. |

Returns

The number of characters, single and multibyte, or `0` if `s` is `NULL`, empty, or has invalid multibyte sequences.

See also

colr_mb_len

Referenced by _rainbow().

#### 0.6.1.2.137 colr_str_noncode_len()

```
size_t colr_str_noncode_len (
            const char * s )
```

Returns the length of string (`char*`), ignoring escape codes and the the null-terminator.

Parameters

| | | |
|---|---|---|
| in | *s* | String to get the length for.<br>Input *must be `null`-terminated*. |

Returns

The length of the string, as if it didn't contain escape codes. For non-escape-code strings, this is like `strlen()`. For `NULL` or "empty" strings, `0` is returned.

See also

colr_str_strip_codes

Referenced by ColorText_length(), colr_str_center(), colr_str_ljust(), and colr_str_rjust().

#### 0.6.1.2.138 colr_str_replace()

```
char* colr_str_replace (
            const char *restrict s,
            const char *restrict target,
            const char *restrict repl )
```

Replaces the first substring found in a string (`char*`).

Using `NULL` as a replacement is like using an empty string (""), which removes the `target` string from `s`.

For a more dynamic version, see the colr_replace and colr_replace_re macros.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *target* | The string to replace. |
| in | *repl* | The string to replace with. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, or target is NULL/empty.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace_ColorArg(), colr_str_replace_ColorResult(), and colr_str_replace_↩ ColorText().

0.6.1.2.139 colr_str_replace_all()

```
char* colr_str_replace_all (
            const char *restrict s,
            const char *restrict target,
            const char *restrict repl )
```

Replaces the first substring found in a string (char*).

Using NULL as a replacement is like using an empty string (""), which removes the target string from s.

For a more dynamic version, see the colr_replace and colr_replace_re macros.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *target* | The string to replace. |
| in | *repl* | The string to replace with. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, or target is NULL/empty.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace_all_ColorArg(), colr_str_replace_all_ColorResult(), and colr_str_↩ replace_all_ColorText().

#### 0.6.1.2.140 colr_str_replace_all_ColorArg()

```
char* colr_str_replace_all_ColorArg (
            const char *restrict s,
            const char *restrict target,
            ColorArg * repl )
```

Replace all substrings in a string (`char*`) with a ColorArg's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *target* | The string to replace. |
| `in` | *repl* | The ColorArg to produce escape-codes to replace with. ColorArg_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, or `target` is `NULL`/empty.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.1.2.141 colr_str_replace_all_ColorResult()

```
char* colr_str_replace_all_ColorResult (
            const char *restrict s,
            const char *restrict target,
            ColorResult * repl )
```

Replace all substrings in a string (`char*`) with a ColorResult's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *target* | The string to replace. |
| in | *repl* | The ColorResult to produce escape-codes to replace with. ColorResult_free() is called after the replacement is done. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, or target is NULL/empty.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

0.6.1.2.142 colr_str_replace_all_ColorText()

```
char* colr_str_replace_all_ColorText (
            const char *restrict s,
            const char *restrict target,
            ColorText * repl )
```

Replace all substrings in a string (char*) with a ColorText's string result.

Using NULL as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *target* | The string to replace. |
| in | *repl* | The ColorText to produce text/escape-codes to replace with. ColorText_free() is called after the replacement is done. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, or target is NULL/empty.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.1.2.143 colr_str_replace_cnt()

```
char* colr_str_replace_cnt (
            const char *restrict s,
            const char *restrict target,
            const char *restrict repl,
            int count )
```

Replaces one or more substrings in a string (`char*`).

Using `NULL` as a replacement is like using an empty string (""), which removes the `target` string from `s`.

For a more dynamic version, see the colr_replace and colr_replace_re macros.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *target* | The string to replace. |
| `in` | *repl* | The string to replace with. |
| `in` | *count* | Number of substrings to replace, or `0` to replace all substrings. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, or `target` is `NULL`/empty.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace(), and colr_str_replace_all().

#### 0.6.1.2.144 colr_str_replace_ColorArg()

```
char* colr_str_replace_ColorArg (
            const char *restrict s,
            const char *restrict target,
            ColorArg * repl )
```

Replace a substring in a string (`char*`) with a ColorArg's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *target* | The string to replace. |
| `in` | *repl* | The ColorArg to produce escape-codes to replace with. ColorArg_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, or `target` is `NULL`/empty.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

colr_replace
colr_replace_re

#### 0.6.1.2.145 colr_str_replace_ColorResult()

```
char* colr_str_replace_ColorResult (
            const char *restrict s,
            const char *restrict target,
            ColorResult * repl )
```

Replace a substring in a string (`char*`) with a ColorResult's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *target* | The string to replace. |
| `in` | *repl* | The ColorResult to produce escape-codes to replace with. ColorResult_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, or `target` is `NULL`/empty.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

colr_replace
colr_replace_re

#### 0.6.1.2.146 colr_str_replace_ColorText()

```
char* colr_str_replace_ColorText (
            const char *restrict s,
            const char *restrict target,
            ColorText * repl )
```

Replace a substring in a string (`char*`) with a ColorText's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *target* | The string to replace. |
| in | *repl* | The ColorText to produce text/escape-codes to replace with. ColorText_free() is called after the replacement is done. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, or target is NULL/empty.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

0.6.1.2.147 colr_str_replace_re()

```
char* colr_str_replace_re (
            const char *restrict s,
            const char *restrict pattern,
            const char *restrict repl,
            int re_flags )
```

Replaces a substring from a regex pattern string (char*) in a string (char*).

Using NULL as a replacement is like using an empty string (""), which removes the target string from s.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *pattern* | The regex match object to find text to replace. |
| in | *repl* | The string to replace with. |
| in | *re_flags* | Flags for regcomp(). REG_EXTENDED is always used, whether flags are provided or not. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, pattern is NULL, or the regex pattern
doesn't compile/match.
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace_re_ColorArg(), colr_str_replace_re_ColorResult(), and colr_str_↩ replace_re_ColorText().

#### 0.6.1.2.148 colr_str_replace_re_all()

```
char* colr_str_replace_re_all (
            const char *restrict s,
            const char *restrict pattern,
            const char *restrict repl,
            int re_flags )
```

Replaces all substrings from a regex pattern string (`char*`) in a string (`char*`).

Using `NULL` as a replacement is like using an empty string (""), which removes the `target` string from `s`.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *pattern* | The regex match object to find text to replace. |
| `in` | *repl* | The string to replace with. |
| `in` | *re_flags* | Flags for `regcomp()`. `REG_EXTENDED` is always used, whether flags are provided or not. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `pattern` is `NULL`, or the regex pattern
doesn't compile/match.
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace_re_all_ColorArg(), colr_str_replace_re_all_ColorResult(), and colr_↩ str_replace_re_all_ColorText().

#### 0.6.1.2.149 colr_str_replace_re_all_ColorArg()

```
char* colr_str_replace_re_all_ColorArg (
            const char *restrict s,
            const char *restrict pattern,
            ColorArg * repl,
            int re_flags )
```

Replace all substrings from a regex pattern string (`char*`) in a string (`char*`) with a ColorArg's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *pattern* | The regex pattern to compile. |
| `in` | *repl* | The ColorArg to produce escape-codes to replace with. ColorArg_free() is called after the replacement is done. |
| `in` | *re_flags* | Flags for `regcomp()`. `REG_EXTENDED` is always used, whether flags are provided or not. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `pattern` is `NULL`, or the regex pattern doesn't compile/match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

colr_replace
colr_replace_re

#### 0.6.1.2.150 colr_str_replace_re_all_ColorResult()

```
char* colr_str_replace_re_all_ColorResult (
            const char *restrict s,
            const char *restrict pattern,
            ColorResult * repl,
            int re_flags )
```

Replace all substrings from a regex pattern string (`char*`) in a string (`char*`) with a ColorResult's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *pattern* | The regex match object to find text to replace. |
| `in` | *repl* | The ColorResult to produce escape-codes to replace with. ColorResult_free() is called after the replacement is done. |
| `in` | *re_flags* | Flags for `regcomp()`. `REG_EXTENDED` is always used, whether flags are provided or not. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, pattern is NULL, or the regex pattern doesn't compile/match.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.1.2.151 colr_str_replace_re_all_ColorText()

```
char* colr_str_replace_re_all_ColorText (
            const char *restrict s,
            const char *restrict pattern,
            ColorText * repl,
            int re_flags )
```

Replace all substrings from a regex pattern string (char*) in a string (char*) with a ColorText's string result.

Using NULL as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *pattern* | The regex match object to find text to replace. |
| in | *repl* | The ColorText to produce text/escape-codes to replace with. ColorText_free() is called after the replacement is done. |
| in | *re_flags* | Flags for regcomp(). REG_EXTENDED is always used, whether flags are provided or not. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, pattern is NULL, or the regex pattern doesn't compile/match.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.1.2.152 colr_str_replace_re_ColorArg()

```
char* colr_str_replace_re_ColorArg (
            const char *restrict s,
            const char *restrict pattern,
            ColorArg * repl,
            int re_flags )
```

Replace substrings from a regex pattern string (`char*`) in a string (`char*`) with a ColorArg's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *pattern* | The regex pattern to compile. |
| `in` | *repl* | The ColorArg to produce escape-codes to replace with. ColorArg_free() is called after the replacement is done. |
| `in` | *re_flags* | Flags for `regcomp()`. `REG_EXTENDED` is always used, whether flags are provided or not. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `pattern` is `NULL`, or the regex pattern doesn't compile/match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

colr_replace
colr_replace_re

#### 0.6.1.2.153 colr_str_replace_re_ColorResult()

```
char* colr_str_replace_re_ColorResult (
            const char *restrict s,
            const char *restrict pattern,
            ColorResult * repl,
            int re_flags )
```

Replace substrings from a regex pattern string (`char*`) in a string (`char*`) with a ColorResult's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *pattern* | The regex match object to find text to replace. |
| `in` | *repl* | The ColorResult to produce escape-codes to replace with. ColorResult_free() is called after the replacement is done. |
| `in` | *re_flags* | Flags for `regcomp()`. `REG_EXTENDED` is always used, whether flags are provided or not. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `pattern` is `NULL`, or the regex pattern doesn't compile/match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.1.2.154 colr_str_replace_re_ColorText()

```
char* colr_str_replace_re_ColorText (
            const char *restrict s,
            const char *restrict pattern,
            ColorText * repl,
            int re_flags )
```

Replace substrings from a regex pattern string (`char*`) in a string (`char*`) with a ColorText's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *pattern* | The regex match object to find text to replace. |
| `in` | *repl* | The ColorText to produce text/escape-codes to replace with. ColorText_free() is called after the replacement is done. |
| `in` | *re_flags* | Flags for `regcomp()`. `REG_EXTENDED` is always used, whether flags are provided or not. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `pattern` is `NULL`, or the regex pattern doesn't compile/match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.1.2.155 colr_str_replace_re_match()

```
char* colr_str_replace_re_match (
            const char *restrict s,
            regmatch_t * match,
            const char *restrict repl )
```

Replaces substrings from a single regex match (`regmatch_t*`) in a string (`char*`).

Using `NULL` as a replacement is like using an empty string (""), which removes the `target` string from `s`.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *match* | The regex match object to find text to replace. |
| `in` | *repl* | The string to replace with. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `match` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace_re_match_ColorArg(), colr_str_replace_re_match_ColorResult(), colr_str_replace_re_match_ColorText(), and colr_str_replace_re_pat().

#### 0.6.1.2.156 colr_str_replace_re_match_ColorArg()

```
char* colr_str_replace_re_match_ColorArg (
            const char *restrict s,
            regmatch_t * match,
            ColorArg * repl )
```

Replace substrings from a regex match (`regmatch_t*`) in a string (`char*`) with a ColorArg's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *match* | The regex match object to find text to replace. |
| `in` | *repl* | The ColorArg to produce escape-codes to replace with. ColorArg_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `match` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.1.2.157 colr_str_replace_re_match_ColorResult()

```
char* colr_str_replace_re_match_ColorResult (
            const char *restrict s,
            regmatch_t * match,
            ColorResult * repl )
```

Replace substrings from a regex match (`regmatch_t*`) in a string (`char*`) with a ColorResult's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *match* | The regex match object to find text to replace. |
| `in` | *repl* | The ColorResult to produce escape-codes to replace with. ColorResult_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `match` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.1.2.158 colr_str_replace_re_match_ColorText()

```
char* colr_str_replace_re_match_ColorText (
            const char *restrict s,
```

```
            regmatch_t * match,
            ColorText * repl )
```

Replace substrings from a regex match (`regmatch_t*`) in a string (`char*`) with a ColorText's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *match* | The regex match object to find text to replace. |
| `in` | *repl* | The ColorText to produce text/escape-codes to replace with. ColorText_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `match` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.1.2.159 colr_str_replace_re_match_i()

```
char* colr_str_replace_re_match_i (
            const char *restrict ref,
            char * target,
            regmatch_t * match,
            const char *restrict repl )
```

Replaces substrings from a regex match (`regmatch_t*`) in a string (`char*`).

This modifies `target` in place. It must have capacity for the result.

Using `NULL` as a replacement is like using an empty string (""), which removes the `target` string from `s`.

Parameters

| | | |
|---|---|---|
| `in` | *ref* | The string to use for offset references. Can be `target`. Set this to the source string if `target` has not been filled yet. If `target` has been filled, you may use `target` for both `ref` and `target`. |
| `out` | *target* | The string to modify. Must have room for the resulting string. |
| `in` | *match* | The regex match object to find text to replace. |
| `in` | *repl* | The string to replace with. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, match is NULL, or the regex pattern doesn't match.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace_re_matches().

#### 0.6.1.2.160 colr_str_replace_re_matches()

```
char* colr_str_replace_re_matches (
            const char *restrict s,
            regmatch_t ** matches,
            const char *restrict repl )
```

Replaces substrings from an array of regex match (regmatch_t*) in a string (char*).

Using NULL as a replacement is like using an empty string (""), which removes the target string from s.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *matches* | Regex match objects to find text to replace. The array must have NULL as the last member. |
| in | *repl* | The string to replace with. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, match is NULL, or the regex pattern doesn't match.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace_re_matches_ColorArg(), colr_str_replace_re_matches_Color↩Result(), colr_str_replace_re_matches_ColorText(), and colr_str_replace_re_pat_all().

#### 0.6.1.2.161 colr_str_replace_re_matches_ColorArg()

```
char* colr_str_replace_re_matches_ColorArg (
            const char *restrict s,
            regmatch_t ** matches,
            ColorArg * repl )
```

Replace substrings from an array of regex matches (`regmatch_t**`) in a string (`char*`) with a ColorArg's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *matches* | The regex match objects to find text to replace. |
| `in` | *repl* | The ColorArg to produce escape-codes to replace with. ColorArg_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `match` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.1.2.162 colr_str_replace_re_matches_ColorResult()

```
char* colr_str_replace_re_matches_ColorResult (
            const char *restrict s,
            regmatch_t ** matches,
            ColorResult * repl )
```

Replace substrings from an array of regex matches (`regmatch_t**`) in a string (`char*`) with a ColorResult's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *matches* | The regex match objects to find text to replace. |
| `in` | *repl* | The ColorResult to produce escape-codes to replace with. ColorResult_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `match` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.1.2.163 colr_str_replace_re_matches_ColorText()

```
char* colr_str_replace_re_matches_ColorText (
            const char *restrict s,
            regmatch_t ** matches,
            ColorText * repl )
```

Replace substrings from an array of regex matches (`regmatch_t**`) in a string (`char*`) with a ColorText's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *matches* | The regex match objects to find text to replace. |
| `in` | *repl* | The ColorText to produce text/escape-codes to replace with. ColorText_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `match` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.1.2.164 colr_str_replace_re_pat()

```
char* colr_str_replace_re_pat (
            const char *restrict s,
```

```
            regex_t * repattern,
            const char *restrict repl )
```

Replaces regex patterns in a string (`char*`).

Using `NULL` as a replacement is like using an empty string (""), which removes the `target` string from `s`.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *repattern* | The regex pattern to match (`regex_t*`). |
| `in` | *repl* | The string to replace with. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `repattern` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace_re(), colr_str_replace_re_pat_ColorArg(), colr_str_replace_re_pat_↩ ColorResult(), and colr_str_replace_re_pat_ColorText().

#### 0.6.1.2.165 colr_str_replace_re_pat_all()

```
char* colr_str_replace_re_pat_all (
            const char *restrict s,
            regex_t * repattern,
            const char *restrict repl )
```

Replaces all matches to a regex pattern in a string (`char*`).

Using `NULL` as a replacement is like using an empty string (""), which removes the `target` string from `s`.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *repattern* | The regex pattern to match (`regex_t*`). |
| `in` | *repl* | The string to replace with. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `repattern` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace_re_all(), colr_str_replace_re_pat_all_ColorArg(), colr_str_replace_↩ re_pat_all_ColorResult(), and colr_str_replace_re_pat_all_ColorText().

#### 0.6.1.2.166 colr_str_replace_re_pat_all_ColorArg()

```
char* colr_str_replace_re_pat_all_ColorArg (
            const char *restrict s,
            regex_t * repattern,
            ColorArg * repl )
```

Replace all matches to a regex pattern in a string (`char*`) with a ColorArg's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *repattern* | The regex pattern to match (`regex_t*`). |
| `in` | *repl* | The ColorArg to produce escape-codes to replace with. ColorArg_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `repattern` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.1.2.167 colr_str_replace_re_pat_all_ColorResult()

```
char* colr_str_replace_re_pat_all_ColorResult (
            const char *restrict s,
            regex_t * repattern,
            ColorResult * repl )
```

Replace all matches to a regex pattern in a string (`char*`) with a ColorResult's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *repattern* | The regex pattern to match (`regex_t*`). |
| `in` | *repl* | The ColorResult to produce escape-codes to replace with. ColorResult_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `repattern` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

colr_replace
colr_replace_re

#### 0.6.1.2.168 colr_str_replace_re_pat_all_ColorText()

```
char* colr_str_replace_re_pat_all_ColorText (
            const char *restrict s,
            regex_t * repattern,
            ColorText * repl )
```

Replace all matches to a regex pattern in a string (`char*`) with a ColorText's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *repattern* | The regex pattern to match (`regex_t*`). |
| `in` | *repl* | The ColorText to produce text/escape-codes to replace with. ColorText_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `repattern` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.1.2.169 colr_str_replace_re_pat_ColorArg()

```
char* colr_str_replace_re_pat_ColorArg (
            const char *restrict s,
            regex_t * repattern,
            ColorArg * repl )
```

Replace regex patterns in a string (`char*`) with a ColorArg's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *repattern* | The regex pattern to match (`regex_t*`). |
| `in` | *repl* | The ColorArg to produce escape-codes to replace with. ColorArg_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `repattern` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.1.2.170 colr_str_replace_re_pat_ColorResult()

```
char* colr_str_replace_re_pat_ColorResult (
            const char *restrict s,
            regex_t * repattern,
            ColorResult * repl )
```

Replace regex patterns in a string (`char*`) with a ColorResult's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *repattern* | The regex pattern to match (regex_t*). |
| in | *repl* | The ColorResult to produce escape-codes to replace with. ColorResult_free() is called after the replacement is done. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, repattern is NULL, or the regex pattern doesn't match.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned.*

See also

colr_replace
colr_replace_re

#### 0.6.1.2.171 colr_str_replace_re_pat_ColorText()

```
char* colr_str_replace_re_pat_ColorText (
            const char *restrict s,
            regex_t * repattern,
            ColorText * repl )
```

Replace regex patterns in a string (char*) with a ColorText's string result.

Using NULL as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *repattern* | The regex pattern to match (regex_t*). |
| in | *repl* | The ColorText to produce text/escape-codes to replace with. ColorText_free() is called after the replacement is done. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, repattern is NULL, or the regex pattern doesn't match.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned.*

See also

colr_replace
colr_replace_re

#### 0.6.1.2.172 colr_str_repr()

```
char* colr_str_repr (
            const char * s )
```

Convert a string (`char*`) into a representation of a string, by wrapping it in quotes and escaping characters that need escaping.

If `s` is NULL, then an allocated string containing the string "NULL" is returned (without quotes).

Escape codes will be escaped, so the terminal will ignore them if the result is printed.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to represent. |

Returns

An allocated string with the representation.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_char_should_escape
colr_char_escape_char

Referenced by _colr_ptr_repr(), ColorResult_repr(), and ColorText_repr().

#### 0.6.1.2.173 colr_str_rjust()

```
char* colr_str_rjust (
            const char * s,
            int width,
            const char padchar )
```

Right-justifies a string (`char*`), ignoring escape codes when measuring the width.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to justify.<br>Input *must be `null`-terminated*. |
| `in` | *width* | The overall width for the resulting string. If set to '0', the terminal width will be used from colr_term_size(). |
| `in` | *padchar* | The character to pad with. If '0', then '' ' is used. |

Returns

An allocated string with the result, or NULL if s is NULL.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned.*

See also

colr_str_center
colr_str_ljust
colr_term_size

Referenced by colr_printf_handler().

### 0.6.1.2.174 colr_str_starts_with()

```
bool colr_str_starts_with (
            const char *restrict s,
            const char *restrict prefix )
```

Checks a string (char*) for a certain prefix substring.

prefix *Must be null-terminated.*

Parameters

| | | |
|---|---|---|
| in | *s* | The string to check. |
| in | *prefix* | The prefix string to look for. |

Returns

True if the string s starts with prefix.
False if one of the strings is null, or the prefix isn't found.

### 0.6.1.2.175 colr_str_strip_codes()

```
char* colr_str_strip_codes (
            const char * s )
```

Strips escape codes from a string (char*), resulting in a new allocated string.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to strip escape codes from.<br>Input *must be null-terminated.* |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_str_noncode_len

Referenced by colr_printf_handler().

#### 0.6.1.2.176 colr_str_to_lower()

```
char* colr_str_to_lower (
            const char * s )
```

Allocate a new lowercase version of a string (`char*`).

*You must `free()` the memory allocated by this function.*

Parameters

| | | |
|---|---|---|
| `in` | *s* | The input string to convert to lower case.<br>*Must be `null`-terminated*. |

Returns

The allocated string, or `NULL` if `s` is `NULL`.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

Referenced by ExtendedValue_from_str(), and RGB_from_str().

#### 0.6.1.2.177 colr_supports_rgb()

```
bool colr_supports_rgb (
            void  )
```

Determine whether the current environment support RGB (True Colors).

This checks `$COLORTERM` for the appropriate value (`'truecolor'` or `'24bit'`). On "dumber" terminals, RGB codes are probably ignored or mistaken for a 256-color or even 8-color value.

For instance, RGB is supported in `konsole`, but not in `xterm` or `linux` ttys. Using RGB codes in `xterm` makes the colors appear as though a 256-color value was used (closest matching value, like RGB_to_term_RGB()). Using RGB codes in a simpler `linux` tty makes them appear as though an 8-color value was used. Very ugly, but not a disaster.

I haven't seen a *modern* linux terminal spew garbage across the screen from using RGB codes when they are not supported, but I could be wrong. I would like to see that terminal if you know of one.

Returns

`true` if 24-bit (true color, or "rgb") support is detected, otherwise `false`.

Referenced by colr_supports_rgb_static().

#### 0.6.1.2.178 colr_supports_rgb_static()

```
bool colr_supports_rgb_static (
            void  )
```

Same as colr_supports_rgb(), but the environment is only checked on the first call.

All other calls return the same result as the first call.

Returns

`true` if 24-bit (true color, or "rgb") support is detected, otherwise `false`.

#### 0.6.1.2.179 colr_term_size()

```
TermSize colr_term_size (
            void  )
```

Attempts to retrieve the row/column size of the terminal and returns a TermSize.

If the call fails, the environment variables `LINES` and `COLUMNS` are checked. If that fails, a default `TermSize` struct is returned:

```
(TermSize){.rows=35, .columns=80}
```

Returns

A TermSize struct with terminal size information.

Referenced by ColorText_length(), colr_str_center(), colr_str_ljust(), and colr_str_rjust().

#### 0.6.1.2.180 colr_win_size()

```
struct winsize colr_win_size (
            void  )
```

Attempts to retrieve a `winsize` struct from an `ioctl` call.

If the call fails, the environment variables `LINES` and `COLUMNS` are checked. If that fails, a default `winsize` struct is returned:

```
(struct winsize){.ws_row=35, .ws_col=80, .ws_xpixel=0, .ws_ypixel=0}
```

`man ioctl_tty` says that `.ws_xpixel` and `.ws_ypixel` are unused.

Returns

A `winsize` struct (`sys/ioctl.h`) with window size information.

Referenced by colr_term_size().

#### 0.6.1.2.181 colr_win_size_env()

```
struct winsize colr_win_size_env (
            void  )
```

Get window/terminal size using the environment variables `LINES`, `COLUMNS`, or `COLS`.

This is used as a fallback if the `ioctl()` call fails in colr_win_size(). If environment variables are not available, a default `winsize` struct is returned:

```
(struct winsize){.ws_row=35, .ws_col=80, .ws_xpixel=0, .ws_ypixel=0}
```

Returns

A `winsize` struct (`sys/ioctl.h`) with window size information.

Referenced by colr_win_size().

#### 0.6.1.2.182 ExtendedValue_eq()

```
bool ExtendedValue_eq (
            ExtendedValue a,
            ExtendedValue b )
```

Compares two ExtendedValues.

This is used to implement colr_eq().

Parameters

| | | |
|---|---|---|
| in | *a* | The first ExtendedValue to compare. |
| in | *b* | The second ExtendedValue to compare. |

Returns

`true` if they are equal, otherwise `false`.

See also

ExtendedValue

0.6.1.2.183 ExtendedValue_from_BasicValue()

```
int ExtendedValue_from_BasicValue (
            BasicValue bval )
```

Convert a BasicValue into an ExtendedValue.

BASIC_INVALID, and other invalid BasicValues will return EXT_INVALID.

Parameters

| | | |
|---|---|---|
| in | *bval* | BasicValue to convert. |

Returns

An ExtendedValue `0–15` on success, otherwise EXT_INVALID.

See also

ExtendedValue

0.6.1.2.184 ExtendedValue_from_esc()

```
int ExtendedValue_from_esc (
            const char * s )
```

Convert an escape-code string (`char*`) to an ExtendedValue.

Parameters

| | | |
|---|---|---|
| in | *s* | Escape-code string.<br>*Must be null-terminated.* |

Return values

| | |
|---|---|
| *An* | integer in the range 0–255 on success. |
| *EXT_INVALID* | on error (or if s is NULL). |
| *EXT_INVALID_RANGE* | if the code number was outside of the range 0–255. |

See also

ExtendedValue

#### 0.6.1.2.185 ExtendedValue_from_hex()

```
int ExtendedValue_from_hex (
            const char * hexstr )
```

Create an ExtendedValue from a hex string (char*).

This is not a 1:1 translation of hex to rgb. Use RGB_from_hex() for that. This will convert the hex string to the closest matching ExtendedValue value.

The format for hex strings can be one of:

- "[#]ffffff" (Leading hash symbol is optional)
- "[#]fff" (short-form)

Parameters

| | | |
|---|---|---|
| in | *hexstr* | Hex string to convert. |

Returns

A value between 0 and 255 on success.

Return values

| | |
|---|---|
| *COLOR_INVALID* | on error or bad values. |

See also

ExtendedValue

Referenced by ExtendedValue_from_hex_default(), and ExtendedValue_from_str().

#### 0.6.1.2.186 ExtendedValue_from_hex_default()

```
ExtendedValue ExtendedValue_from_hex_default (
            const char * hexstr,
            ExtendedValue default_value )
```

Create an ExtendedValue from a hex string (`char*`), but return a default value if the hex string is invalid.

This is not a 1:1 translation of hex to rgb. Use RGB_from_hex_default() for that. This will convert the hex string to the closest matching ExtendedValue value.

The format for hex strings can be one of:

- "[#]ffffff" (Leading hash symbol is optional)
- "[#]fff" (short-form)

Parameters

| | | |
|---|---|---|
| `in` | *hexstr* | Hex string to convert. |
| `in` | *default_value* | ExtendedValue to use for bad hex strings. |

Returns

An ExtendedValue on success, or `default_value` on error.

See also

ExtendedValue
ExtendedValue_from_hex

#### 0.6.1.2.187 ExtendedValue_from_RGB()

```
ExtendedValue ExtendedValue_from_RGB (
            RGB rgb )
```

Convert an RGB value into the closest matching ExtendedValue.

Parameters

| | | |
|---|---|---|
| in | *rgb* | RGB value to convert. |

Returns

An ExtendedValue that closely matches the original RGB value.

See also

ExtendedValue

Referenced by ExtendedValue_from_hex(), format_bg_RGB_term(), and format_fg_RGB_term().

#### 0.6.1.2.188 ExtendedValue_from_str()

```
int ExtendedValue_from_str (
            const char * arg )
```

Converts a known name, integer string (0-255), or a hex string (`char*`), into an ExtendedValue suitable for the extended-value-based functions.

Hex strings can be used:

- "#ffffff" (Leading hash symbol is **NOT** optional)
- "#fff" (short-form)

The "#" is not optional for hex strings because it is impossible to tell the difference between the hex value '111' and the extended value '111' without it.

Parameters

| | | |
|---|---|---|
| in | *arg* | Color name to find the ExtendedValue for. |

Returns

A value between 0 and 255 on success.

Return values

| | |
|---|---|
| *EXT_INVALID* | on error or bad values. |
| *EXT_INVALID_RANGE* | if the number was outside of the range `0-255`. |

See also

ExtendedValue

#### 0.6.1.2.189 ExtendedValue_is_invalid()

```
bool ExtendedValue_is_invalid (
            int eval )
```

Determines whether an integer is an invalid ExtendedValue.

Parameters

| | | |
|---|---|---|
| in | *eval* | A number to check. |

Returns

`true` if the value is considered invalid, otherwise `false`.

See also

ExtendedValue

#### 0.6.1.2.190 ExtendedValue_is_valid()

```
bool ExtendedValue_is_valid (
            int eval )
```

Determines whether an integer is a valid ExtendedValue.

Parameters

| | | |
|---|---|---|
| in | *eval* | A number to check. |

Returns

`true` if the value is considered valid, otherwise `false`.

See also

ExtendedValue

#### 0.6.1.2.191 ExtendedValue_repr()

```
char* ExtendedValue_repr (
            int eval )
```

Creates a string (`char*`) representation of a ExtendedValue.

Parameters

| | | |
|---|---|---|
| in | *eval* | A ExtendedValue to get the value from. |

Returns

A pointer to an allocated string.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

ExtendedValue

#### 0.6.1.2.192 ExtendedValue_to_str()

```
char* ExtendedValue_to_str (
            ExtendedValue eval )
```

Creates a human-friendly string (`char*`) from an ExtendedValue's actual value, suitable for use with ExtendedValue_from_str().

Parameters

| | | |
|---|---|---|
| in | *eval* | A ExtendedValue to get the value from. |

Returns

A pointer to an allocated string
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

ExtendedValue

#### 0.6.1.2.193 format_bg()

```
void format_bg (
            char * out,
            BasicValue value )
```

Create an escape code for a background color.

Parameters

| | | |
|---|---|---|
| out | *out* | Memory allocated for the escape code string. *Must have enough room for CODEX_LEN*. |
| in | *value* | BasicValue value to use for background. |

#### 0.6.1.2.194 format_bg_RGB()

```
void format_bg_RGB (
            char * out,
            RGB rgb )
```

Create an escape code for a true color (rgb) background color using values from an RGB struct.

Parameters

| | | |
|---|---|---|
| out | *out* | Memory allocated for the escape code string. *Must have enough room for CODE_RGB_LEN*. |
| in | *rgb* | RGB struct to get red, blue, and green values from. |

Referenced by _rainbow(), and rainbow_bg().

#### 0.6.1.2.195 format_bg_RGB_term()

```
void format_bg_RGB_term (
            char * out,
            RGB rgb )
```

Create an escape code for a true color (rgb) fore color using an RGB struct's values, approximating 256-color values.

Parameters

| | | |
|---|---|---|
| out | *out* | Memory allocated for the escape code string. |
| in | *rgb* | Pointer to an RGB struct. |

Referenced by _rainbow(), and rainbow_bg_term().

#### 0.6.1.2.196 format_bgx()

```
void format_bgx (
            char * out,
            unsigned char num )
```

Create an escape code for an extended background color.

Parameters

| | | |
|---|---|---|
| out | *out* | Memory allocated for the escape code string. *Must have enough room for CODEX_LEN*. |
| in | *num* | Value to use for background. |

Referenced by format_bg_RGB_term().

#### 0.6.1.2.197 format_fg()

```
void format_fg (
            char * out,
            BasicValue value )
```

Create an escape code for a fore color.

Parameters

| | | |
|---|---|---|
| out | *out* | Memory allocated for the escape code string. *Must have enough room for CODEX_LEN*. |
| in | *value* | BasicValue value to use for fore. |

#### 0.6.1.2.198 format_fg_RGB()

```
void format_fg_RGB (
            char * out,
            RGB rgb )
```

Create an escape code for a true color (rgb) fore color using an RGB struct's values.

Parameters

| | | |
|---|---|---|
| out | *out* | Memory allocated for the escape code string. |
| in | *rgb* | Pointer to an RGB struct. |

Referenced by rainbow_fg().

#### 0.6.1.2.199 format_fg_RGB_term()

```
void format_fg_RGB_term (
            char * out,
            RGB rgb )
```

Create an escape code for a true color (rgb) fore color using an RGB struct's values, approximating 256-color values.

Parameters

| | | |
|---|---|---|
| out | *out* | Memory allocated for the escape code string. |
| in | *rgb* | Pointer to an RGB struct. |

Referenced by rainbow_fg_term().

#### 0.6.1.2.200 format_fgx()

```
void format_fgx (
            char * out,
            unsigned char num )
```

Create an escape code for an extended fore color.

Parameters

| | | |
|---|---|---|
| out | *out* | Memory allocated for the escape code string. *Must have enough room for CODEX_LEN*. |
| in | *num* | Value to use for fore. |

Referenced by format_fg_RGB_term().

#### 0.6.1.2.201 format_style()

```
void format_style (
            char * out,
            StyleValue style )
```

Create an escape code for a style.

Parameters

| | | |
|---|---|---|
| out | *out* | Memory allocated for the escape code string. *Must have enough room for STYLE_LEN*. |
| in | *style* | StyleValue value to use for style. |

#### 0.6.1.2.202 rainbow_bg()

```
char* rainbow_bg (
            const char * s,
            double freq,
```

```
            size_t offset,
            size_t spread )
```

Rainbow-ize some text using rgb back colors, lolcat style.

This prepends a color code to every character in the string. Multi-byte characters are handled properly by checking colr_mb_len(), and copying the bytes to the resulting string without codes between the multibyte characters.

The CODE_RESET_ALL code is appended to the result.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to colorize.<br>Input *must be null-terminated*. |
| in | *freq* | Frequency ("tightness") for the colors. |
| in | *offset* | Starting offset in the rainbow. |
| in | *spread* | Number of characters per color. |

Returns

The allocated/formatted string on success.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

#### 0.6.1.2.203 rainbow_bg_term()

```
char* rainbow_bg_term (
            const char * s,
            double freq,
            size_t offset,
            size_t spread )
```

This is exactly like rainbow_bg(), except it uses colors that are closer to the standard 256-color values.

This prepends a color code to every character in the string. Multi-byte characters are handled properly by checking colr_mb_len(), and copying the bytes to the resulting string without codes between the multibyte characters.

The CODE_RESET_ALL code is appended to the result.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to colorize.<br>Input *must be null-terminated*. |
| in | *freq* | Frequency ("tightness") for the colors. |
| in | *offset* | Starting offset in the rainbow. |
| in | *spread* | Number of characters per color. |

Returns

The allocated/formatted string on success.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

#### 0.6.1.2.204 rainbow_fg()

```
char* rainbow_fg (
            const char * s,
            double freq,
            size_t offset,
            size_t spread )
```

Rainbow-ize some text using rgb fore colors, lolcat style.

This prepends a color code to every character in the string. Multi-byte characters are handled properly by checking colr_mb_len(), and copying the bytes to the resulting string without codes between the multibyte characters.

The `CODE_RESET_ALL` code is appended to the result.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to colorize.<br>Input *must be `null`-terminated*. |
| `in` | *freq* | Frequency ("tightness") for the colors. |
| `in` | *offset* | Starting offset in the rainbow. |
| `in` | *spread* | Number of characters per color. |

Returns

The allocated/formatted string on success.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

#### 0.6.1.2.205 rainbow_fg_term()

```
char* rainbow_fg_term (
            const char * s,
            double freq,
            size_t offset,
            size_t spread )
```

This is exactly like rainbow_fg(), except it uses colors that are closer to the standard 256-color values.

This prepends a color code to every character in the string. Multi-byte characters are handled properly by checking colr_mb_len(), and copying the bytes to the resulting string without codes between the multibyte characters.

The `CODE_RESET_ALL` code is appended to the result.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to colorize.<br>Input *must be `null`-terminated*. |
| in | *freq* | Frequency ("tightness") for the colors. |
| in | *offset* | Starting offset in the rainbow. |
| in | *spread* | Number of characters per color. |

Returns

The allocated/formatted string on success.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

#### 0.6.1.2.206 rainbow_step()

```
RGB rainbow_step (
            double freq,
            size_t offset )
```

A single step in rainbow-izing produces the next color in the "rainbow" as an RGB value.

Parameters

| | | |
|---|---|---|
| in | *freq* | Frequency ("tightness") of the colors. |
| in | *offset* | Starting offset in the rainbow. |

Returns

An RGB value with the next "step" in the "rainbow".

Referenced by _rainbow().

#### 0.6.1.2.207 RGB_average()

```
unsigned char RGB_average (
            RGB rgb )
```

Return the average for an RGB value.

This is also it's "grayscale" value.

Parameters

| | | |
|---|---|---|
| in | *rgb* | The RGB value to get the average for. |

Returns

A value between `0-255`.

See also

RGB

Referenced by RGB_grayscale().

#### 0.6.1.2.208 RGB_eq()

```
bool RGB_eq (
            RGB a,
            RGB b )
```

Compare two RGB structs.

Parameters

| | | |
|---|---|---|
| in | *a* | First RGB value to check. |
| in | *b* | Second RGB value to check. |

Returns

`true` if a and b have the same `r`, g, and b values, otherwise `false`.

See also

RGB

Referenced by ColorValue_eq(), and ExtendedValue_from_RGB().

#### 0.6.1.2.209 RGB_from_BasicValue()

```
RGB RGB_from_BasicValue (
            BasicValue bval )
```

Return an RGB value from a known BasicValue.

Terminals use different values to render basic 3/4-bit escape-codes. The values returned from this function match the names found in `colr_name_data[]`.

Parameters

| | | |
|---|---|---|
| in | *bval* | A BasicValue to get the RGB value for. |

Returns

An RGB value that matches the BasicValue's color.

See also

RGB

#### 0.6.1.2.210 RGB_from_esc()

```
int RGB_from_esc (
            const char * s,
            RGB * rgb )
```

Convert an escape-code string (`char*`) to an actual RGB value.

Parameters

| | | |
|---|---|---|
| `in` | *s* | Escape-code string.<br>*Must be `null`-terminated.* |
| `out` | *rgb* | Pointer to an RGB struct to fill in the values for. |

Return values

| | |
|---|---|
| *<tt>0</tt>* | on success, with `rgb` filled with values. |
| *COLOR_INVALID* | on error (or if `s` is `NULL`). |
| *COLOR_INVALID_RANGE* | if any code numbers were outside of the range `0-255`. |

See also

RGB

#### 0.6.1.2.211 RGB_from_ExtendedValue()

```
RGB RGB_from_ExtendedValue (
            ExtendedValue eval )
```

Return an RGB value from a known ExtendedValue.

This is just a type/bounds-checked alias for `ext2rgb_map[eval]`.

Parameters

| | | |
|---|---|---|
| `in` | *eval* | An ExtendedValue to get the RGB value for. |

Returns

An RGB value from `ext2rgb_map[]`.

See also

RGB

#### 0.6.1.2.212 RGB_from_hex()

```
int RGB_from_hex (
            const char * hexstr,
            RGB * rgb )
```

Convert a hex color into an RGB value.

The format for hex strings can be one of:

- "[#]ffffff" (Leading hash symbol is optional)
- "[#]fff" (short-form)

Parameters

| | | |
|---|---|---|
| `in` | *hexstr* | String to check for hex values.<br>Input *must be `null`-terminated*. |
| `out` | *rgb* | Pointer to an RGB struct to fill in the values for. |

Return values

| | |
|---|---|
| *0* | on success, with `rgb` filled with the values. |
| *COLOR_INVALID* | for non-hex strings. |

See also

RGB

Referenced by ExtendedValue_from_hex(), RGB_from_hex_default(), and RGB_from_str().

#### 0.6.1.2.213 RGB_from_hex_default()

```
RGB RGB_from_hex_default (
            const char * hexstr,
            RGB default_value )
```

Convert a hex color into an RGB value, but use a default value when errors occur.

The format for hex strings can be one of:

- "[#]ffffff" (Leading hash symbol is optional)
- "[#]fff" (short-form)

Parameters

| | | |
|---|---|---|
| in | *hexstr* | String to check for RGB values.<br>Input *must be null-terminated*. |
| out | *default_value* | An RGB value to use when errors occur. |

Returns

A valid RGB value on success, or default_value on error.

See also

RGB
hex

0.6.1.2.214 RGB_from_str()

```
int RGB_from_str (
            const char * arg,
            RGB * rgb )
```

Convert an RGB string (char*) into an RGB value.

The format for RGB strings can be one of:

- "RED,GREEN,BLUE"
- "RED GREEN BLUE"
- "RED:GREEN:BLUE"
- "RED;GREEN;BLUE" Or hex strings can be used:
- "#ffffff" (Leading hash symbol is **NOT** optional)
- "#fff" (short-form)

Parameters

| | | |
|---|---|---|
| in | *arg* | String to check for RGB values.<br>Input *must be null-terminated*. |
| out | *rgb* | Pointer to an RGB struct to fill in the values for. |

Return values

| | |
|---|---|
| *0* | on success, with `rgb` filled with the values. |
| *COLOR_INVALID* | for non-rgb strings. |
| *COLOR_INVALID_RANGE* | for rgb values outside of 0-255. |

See also

RGB

#### 0.6.1.2.215 RGB_grayscale()

```
RGB RGB_grayscale (
            RGB rgb )
```

Return a grayscale version of an RGB value.

Parameters

| | | |
|---|---|---|
| `in` | *rgb* | The RGB value to convert. |

Returns

A grayscale RGB value.

See also

RGB

#### 0.6.1.2.216 RGB_inverted()

```
RGB RGB_inverted (
            RGB rgb )
```

Make a copy of an RGB value, with the colors "inverted" (like highlighting text in the terminal).

Parameters

| | | |
|---|---|---|
| `in` | *rgb* | The RGB value to invert. |

Returns

An "inverted" RGB value.

See also

RGB

#### 0.6.1.2.217 RGB_monochrome()

```
RGB RGB_monochrome (
            RGB rgb )
```

Convert an RGB value into either black or white, depending on it's average grayscale value.

Parameters

| | | |
|---|---|---|
| in | *rgb* | The RGB value to convert. |

Returns

Either `rgb(1, 1, 1)` or `rgb(255, 255, 255)`.

See also

RGB

#### 0.6.1.2.218 RGB_repr()

```
char* RGB_repr (
            RGB rgb )
```

Creates a string (`char*`) representation for an RGB value.

Allocates memory for the string representation.

Parameters

| | | |
|---|---|---|
| in | *rgb* | RGB struct to get the representation for. |

Returns

Allocated string for the representation.
*You must `free()` the memory allocated by this function.*

See also

RGB

#### 0.6.1.2.219 RGB_to_hex()

```
char* RGB_to_hex (
            RGB rgb )
```

Converts an RGB value into a hex string (`char*`).

Parameters

| | | |
|---|---|---|
| `in` | *rgb* | RGB value to convert. |

Returns

An allocated string.

*You must `free()` the memory allocated by this function.*

*If allocation fails, `NULL` is returned.*

See also

RGB

#### 0.6.1.2.220 RGB_to_str()

```
char* RGB_to_str (
            RGB rgb )
```

Convert an RGB value into a human-friendly RGB string (`char*`) suitable for input to RGB_from_str().

Parameters

| | | |
|---|---|---|
| `in` | *rgb* | RGB value to convert. |

Returns

An allocated string in the form `"red;green;blue"`.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

RGB

#### 0.6.1.2.221 RGB_to_term_RGB()

```
RGB RGB_to_term_RGB (
            RGB rgb )
```

Convert an RGB value into it's nearest terminal-friendly RGB value.

This is a helper for the 'to_term' functions.

Parameters

| | | |
|---|---|---|
| in | *rgb* | RGB to convert. |

Returns

A new RGB with values close to a terminal code color.

See also

RGB

Referenced by ExtendedValue_from_RGB().

#### 0.6.1.2.222 StyleValue_eq()

```
bool StyleValue_eq (
            StyleValue a,
            StyleValue b )
```

Compares two StyleValues.

This is used to implement colr_eq().

Parameters

| | | |
|---|---|---|
| in | *a* | The first StyleValue to compare. |
| in | *b* | The second StyleValue to compare. |

Returns

`true` if they are equal, otherwise `false`.

See also

StyleValue

0.6.1.2.223 StyleValue_from_esc()

```
StyleValue StyleValue_from_esc (
            const char * s )
```

Convert an escape-code string (`char*`) to an actual StyleValue enum value.

Parameters

| | | |
|---|---|---|
| in | *s* | Escape-code string.<br>*Must be `null`-terminated.* |

Return values

| | |
|---|---|
| *StyleValue* | value on success. |
| *STYLE_INVALID* | on error (or if `s` is `NULL`). |
| *STYLE_INVALID_RANGE* | if the code number was outside of the range `0-255`. |

See also

StyleValue

0.6.1.2.224 StyleValue_from_str()

```
StyleValue StyleValue_from_str (
            const char * arg )
```

Convert a named argument to actual StyleValue enum value.

Parameters

| | | |
|---|---|---|
| in | *arg* | Style name to convert into a StyleValue. |

Returns

A usable StyleValue value on success, or STYLE_INVALID on error.

See also

StyleValue

0.6.1.2.225 StyleValue_is_invalid()

```
bool StyleValue_is_invalid (
            StyleValue sval )
```

Determines whether a StyleValue is invalid.

Parameters

| | | |
|---|---|---|
| in | *sval* | A StyleValue to check. |

Returns

`true` if the value is considered invalid, otherwise `false`.

See also

StyleValue

#### 0.6.1.2.226 StyleValue_is_valid()

```
bool StyleValue_is_valid (
            StyleValue sval )
```

Determines whether a StyleValue is valid.

Parameters

| | | |
|---|---|---|
| in | *sval* | A StyleValue to check. |

Returns

`true` if the value is considered valid, otherwise `false`.

See also

StyleValue

#### 0.6.1.2.227 StyleValue_repr()

```
char* StyleValue_repr (
            StyleValue sval )
```

Creates a string (`char*`) representation of a StyleValue.

Parameters

| | | |
|---|---|---|
| in | *sval* | A StyleValue to get the value from. |

Returns

A pointer to an allocated string.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

StyleValue

#### 0.6.1.2.228 StyleValue_to_str()

```
char* StyleValue_to_str (
            StyleValue sval )
```

Create a human-friendly string (`char*`) representation for a StyleValue.

Parameters

| | | |
|---|---|---|
| `in` | *sval* | StyleValue to get the name for. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

StyleValue

#### 0.6.1.2.229 TermSize_repr()

```
char* TermSize_repr (
            TermSize ts )
```

Create a string (`char*`) representation for a TermSize.

Parameters

| | | |
|---|---|---|
| `in` | *ts* | TermSize to get the representation for. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

TermSize

### 0.6.1.3 Variable Documentation

#### 0.6.1.3.1 basic_names

```
const BasicInfo basic_names[]
```

**Initial value:**

```
= {
    {"reset", RESET},
    {"none", RESET},
    {"black", BLACK},
    {"blue", BLUE},
    {"cyan", CYAN},
    {"green", GREEN},
    {"magenta", MAGENTA},
    {"red", RED},
    {"white", WHITE},
    {"normal", WHITE},
    {"yellow", YELLOW},
    {"lightblack", LIGHTBLACK},
    {"lightblue", LIGHTBLUE},
    {"lightcyan", LIGHTCYAN},
    {"lightgreen", LIGHTGREEN},
    {"lightmagenta", LIGHTMAGENTA},
    {"lightred", LIGHTRED},
    {"lightwhite", LIGHTWHITE},
    {"lightnormal", LIGHTWHITE},
    {"lightyellow", LIGHTYELLOW},
}
```

An array of BasicInfo items, used with BasicValue_from_str().

#### 0.6.1.3.2 colr_printf_esc_mod

```
int colr_printf_esc_mod = 0
```

Integer to test for the presence of the "escaped output modifier" in colr_printf_handler.

This is set in colr_printf_register.

Referenced by colr_printf_handler(), and colr_printf_register().

#### 0.6.1.3.3 ext2rgb_map

```
const RGB ext2rgb_map[]
```

A map from ExtendedValue (256-color) to RGB value, where the index is the is the ExtendedValue, and the value is the RGB.

This is used in several RGB/ExtendedValue functions.

See also

ExtendedValue_from_RGB
RGB_to_term_RGB

#### 0.6.1.3.4 extended_names

```
const ExtendedInfo extended_names[]
```

**Initial value:**

```
= {
    {"xred", XRED},
    {"xgreen", XGREEN},
    {"xyellow", XYELLOW},
    {"xblue", XBLUE},
    {"xmagenta", XMAGENTA},
    {"xcyan", XCYAN},
    {"xwhite", XWHITE},
    {"xnormal", XWHITE},
    {"xlightred", XLIGHTRED},
    {"xlightgreen", XLIGHTGREEN},
    {"xlightyellow", XLIGHTYELLOW},
    {"xlightblack", XLIGHTBLACK},
    {"xlightblue", XLIGHTBLUE},
    {"xlightmagenta", XLIGHTMAGENTA},
    {"xlightwhite", XLIGHTWHITE},
    {"xlightnormal", XLIGHTWHITE},
    {"xlightcyan", XLIGHTCYAN},
}
```

An array of ExtendedInfo, used with ExtendedValue_from_str().

#### 0.6.1.3.5 style_names

```
const StyleInfo style_names[]
```

**Initial value:**

```
= {

    {"reset", RESET_ALL},
    {"none", RESET_ALL},
    {"resetall", RESET_ALL},
    {"reset-all", RESET_ALL},
    {"reset_all", RESET_ALL},
    {"bold", BRIGHT},
    {"bright", BRIGHT},
    {"dim", DIM},
    {"italic", ITALIC},
    {"underline", UNDERLINE},
    {"flash", FLASH},
    {"highlight", HIGHLIGHT},
    {"normal", NORMAL},
    {"strikethru", STRIKETHRU},
    {"strike", STRIKETHRU},
    {"strikethrough", STRIKETHRU},
    {"frame", FRAME},
    {"encircle", ENCIRCLE},
    {"circle", ENCIRCLE},
    {"overline", OVERLINE},
}
```

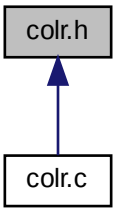
An array of StyleInfo items, used with StyleName_from_str().

### 0.6.2 colr.h File Reference

Declarations for ColrC functions, enums, structs, etc.

```
#include <assert.h>
#include <ctype.h>
#include <math.h>
#include <limits.h>
#include <locale.h>
#include <printf.h>
#include <regex.h>
#include <stdarg.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <unistd.h>
```

`#include <wchar.h>`
This graph shows which files directly or indirectly include this file:




Data Structures

- struct BasicInfo

  *Holds a known color name and it's `BasicValue`. More...*
- struct ColorArg

  *Holds an ArgType, and a ColorValue. More...*
- struct ColorJustify

  *Holds a string justification method, width, and padding character for ColorTexts. More...*
- struct ColorNameData

  *Holds info about a known color name, like it's ExtendedValue and it's RGB value. More...*
- struct ColorResult

  *Holds a string (`char*`) that was definitely allocated by Colr. More...*
- union ColorStructMarker

  *Breaks down Colr struct markers, such as COLORARG_MARKER, into individual bytes. More...*
- struct ColorStructMarker.bytes

  *Individual bytes that make up the marker. More...*
- struct ColorText

  *Holds a string of text, and optional fore, back, and style ColorArgs. More...*
- struct ColorValue

  *Holds a color type and it's value. More...*
- struct ExtendedInfo

  *Holds a known color name and it's `ExtendedValue`. More...*
- struct RGB

  *Container for RGB values. More...*
- struct StyleInfo

  *Holds a known style name and it's `StyleValue`. More...*
- struct TermSize

  *Holds a terminal size, usually retrieved with colr_term_size(). More...*

Macros

- #define alloc_basic() calloc(CODE_LEN, sizeof(char))

  *Allocate enough for a basic code.*
- #define alloc_extended() calloc(CODEX_LEN, sizeof(char))

  *Allocate enough for an extended code.*
- #define alloc_rgb() calloc(CODE_RGB_LEN, sizeof(char))

  *Allocate enough for an rgb code.*
- #define alloc_style() calloc(STYLE_LEN, sizeof(char))

  *Allocate enough for a style code.*
- #define asprintf_or_return(retval, ...) if_not_asprintf(__VA_ARGS__) return retval

  *Convenience macro for bailing out of a function when asprintf fails.*
- #define back(x) ColorArg_to_ptr(back_arg(x))

  *Create a back color suitable for use with the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros.*
- #define back_arg(x)

  *Uses ColorArg_from_<type> to build a ColorArg with the appropriate color type, based on the type of it's argument.*
- #define back_str(x) ColorArg_to_esc(back_arg(x))

  *Return just the escape code string for a back color.*
- #define back_str_static(x)

  *Creates a stack-allocated escape code string (char*) for a back color.*
- #define basic(x) ((BasicValue)(x))

  *Casts to BasicValue.*
- #define bool_colr_enum(x) (x < 0 ? false: true)

  *Returns the "truthiness" of the enums used in ColrC (BasicValue, ExtendedValue function-returns, Style↩ Value, ColorType, ArgType).*
- #define CODE_ANY_LEN 46

  *Maximum length in chars for any possible escape code mixture for one complete style (one of each: fore, back, and style).*
- #define CODE_LEN 14

  *Maximum length for a basic fore/back escape code, including "\0".*
- #define CODE_LEN_MIN 5

  *Minimum length for the shortest basic fore/back escape code, including "\0".*
- #define CODE_RESET_ALL "\x1b[0m"

  *Convenience definition, because this is used a lot.*
- #define CODE_RESET_BACK "\x1b[49m"

  *Convenience definition for resetting the back color.*
- #define CODE_RESET_FORE "\x1b[39m"

  *Convenience definition for resetting the fore color.*
- #define CODE_RESET_LEN 5

  *Length of CODE_RESET_ALL, including "\0".*
- #define CODE_RGB_LEN 20

  *Maximum length in chars for an RGB fore/back escape code, including "\0".*
- #define CODE_RGB_LEN_MIN 14

  *Minimum length for the shortest RGB fore/back escape code, including "\0".*
- #define CODEX_LEN 12

  *Maximum length for an extended fore/back escape code, including "\0".*
- #define CODEX_LEN_MIN 10

  *Minimum length for the shortest extended fore/back escape code, including "\0".*
- #define color_arg(type, x)

  *Builds a correct ColorArg struct according to the type of it's second argument.*

- #define COLOR_INVALID (-2)

  *Possible error return value for BasicValue_from_str(), ExtendedValue_from_str(), and colorname_to_rgb().*
- #define COLOR_INVALID_RANGE (-3)

  *Possible error return value for RGB_from_str().*
- #define COLOR_LEN 30

  *Maximum length in chars for any combination of basic/extended escape codes for one complete style (one of each: fore, back, style).*
- #define color_name_is_invalid(x) ColorType_is_invalid(ColorType_from_str(x))

  *Convenience macro for checking if a color name is invalid.*
- #define color_name_is_valid(x) ColorType_is_valid(ColorType_from_str(x))

  *Convenience macro for checking if a color name is valid.*
- #define COLOR_RGB_LEN 26

  *Maximum length in chars added to a rgb colorized string.*
- #define color_val(x)

  *Builds a correct ColorValue struct according to the type of it's first argument.*
- #define COLORARG_MARKER UINT32_MAX

  *Marker for the ColorArg struct, for identifying a void pointer as a ColorArg.*
- #define COLORJUSTIFY_MARKER (UINT32_MAX - 30)

  *Marker for the ColorJustify struct, for identifying a void pointer as a ColorJustify.*
- #define COLORLASTARG_MARKER (UINT32_MAX - 20)

  *Marker for the _ColrLastArg_s struct, for identifying a void pointer as a _ColrLastArg_s.*
- #define COLORRESULT_MARKER (UINT32_MAX - 40)

  *Marker for the ColorResult struct, for identifying a void pointer as a ColorResult.*
- #define COLORTEXT_MARKER (UINT32_MAX - 50)

  *Marker for the ColorText struct, for identifying a void pointer as a ColorText.*
- #define ColorValue_has(cval, val)

  *Call the current ColorValue_has_<type> function for the given value.*
- #define Colr(text, ...) ColorText_to_ptr(ColorText_from_values(text, __VA_ARGS__, _ColrLast↩ Arg))

  *Returns a heap-allocated ColorText struct that can be used by itself, or with the colr_cat(), colr_join(), Colr↩ _cat(), and Colr_join() macros.*
- #define colr(text, ...) colr_cat(Colr(text, __VA_ARGS__))

  *Create an allocated string directly from Colr() arguments.*
- #define colr_alloc_len(x)

  *Return the number of bytes needed to allocate an escape code string based on the color type.*
- #define colr_asprintf(...) colr_printf_macro(asprintf, __VA_ARGS__)

  *Ensure colr_printf_register() has been called, and then call `asprintf`.*
- #define Colr_cat(...) ColorResult_to_ptr(ColorResult_new(colr_cat(__VA_ARGS__)))

  *Like colr_cat(), but returns an allocated ColorResult that the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros will automatically `free()`.*
- #define colr_cat(...) _colr_join("", __VA_ARGS__, _ColrLastArg)

  *Join ColorArg pointers, ColorResult pointers, ColorText pointers, and strings into one long string.*
- #define Colr_center(text, justwidth, ...)

  *Sets the JustifyMethod for a ColorText while allocating it.*
- #define Colr_center_char(text, justwidth, c, ...)

  *Sets the JustifyMethod for a ColorText while allocating it.*
- #define colr_eq(a, b)

  *Calls the <type>_eq functions for the supported types.*
- #define colr_example(x)

  *Calls the <type>_example functions for the supported types.*
- #define COLR_FMT "R"

*Format character string suitable for use in the printf-family of functions.*

- #define COLR_FMT_CHAR COLR_FMT[0]

  *Character used in printf format strings for Colr objects.*
- #define COLR_FMT_MOD_ESC "/"

  *Modifier for Colr printf character to produce escaped output.*
- #define COLR_FMT_MOD_ESC_CHAR COLR_FMT_MOD_ESC[0]

  *Modifier for Colr printf character to produce escaped output, in char form.*
- #define colr_fprintf(...) colr_printf_macro(fprintf, __VA_ARGS__)

  *Ensure colr_printf_register() has been called, and then call `fprintf`.*
- #define colr_free(x)

  *Calls the <type>_free functions for the supported types.*
- #define COLR_GNU

  *Defined when `__GNUC__` is available, to enable `statement-expressions` and `register_printf_specifier` .*
- #define COLR_HASH_SEED 5381

  *Seed value for colr_str_hash().*
- #define colr_is_empty(x)

  *Calls the <type>is_empty functions for the supported types.*
- #define colr_is_invalid(x)

  *Calls the <type>is_invalid functions for the supported types.*
- #define colr_is_valid(x)

  *Calls the <type>is_valid functions for the supported types.*
- #define colr_is_valid_mblen(x) ((x) && ((x) != (size_t)-1) && ((x) != (size_t)-2))

  *Checks return values from `mbrlen()` and colr_mb_len().*
- #define colr_istr_either(s1, s2, s3)

  *Convenience macro for `!strcasecmp(s1, s2) || !strcasecmp(s1, s3)`.*
- #define colr_istr_eq(s1, s2)

  *Convenience macro for `!strcasecmp(s1, s2)`.*
- #define Colr_join(joiner, ...) ColrResult(colr_join(joiner, __VA_ARGS__))

  *Joins Colr objects and strings, exactly like colr_join(), but returns an allocated ColorResult that the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros will automatically `free()` for you.*
- #define colr_join(joiner, ...) _colr_join(joiner, __VA_ARGS__, _ColrLastArg)

  *Join ColorArg pointers, ColorText pointers, and strings by another ColorArg pointer, ColorText pointer, or string.*
- #define colr_length(x)

  *Calls the <type>_length functions for the supported types.*
- #define Colr_ljust(text, justwidth, ...)

  *Sets the JustifyMethod for a ColorText while allocating it.*
- #define Colr_ljust_char(text, justwidth, c, ...)

  *Sets the JustifyMethod for a ColorText while allocating it.*
- #define colr_max(a, b) (a > b ? a : b)

  *Macro for `(a > b ? a : b)`.*
- #define colr_print(...)

  *Create a string from a colr_cat() call, print it to stdout (without a newline), and free it.*
- #define colr_printf(...) colr_printf_macro(printf, __VA_ARGS__)

  *Ensure colr_printf_register() has been called, and then call `printf`.*
- #define colr_printf_macro(func, ...)

  *Calls one of the printf-family functions, with format warnings disabled for the call, and returns the result.*
- #define colr_puts(...)

  *Create a string from a colr_cat() call, print it (with a newline), and free it.*
- #define colr_replace(s, target, repl)

*Replace a substring in s with another string, ColorArg string, ColorResult string, or ColorText string.*

- #define colr_replace_all(s, target, repl)

  *Replace all substrings in s with another string, ColorArg string, ColorResult string, or ColorText string.*
- #define colr_replace_re(s, target, repl, flags)

  *Replace a regex pattern string (char*) in s with another string, ColorArg string, ColorResult string, or ColorText string.*
- #define colr_replace_re_all(s, target, repl, flags)

  *Replace all matches to a regex pattern string (char*) in s with another string, ColorArg string, ColorResult string, or ColorText string.*
- #define colr_repr(x)

  *Transforms several ColrC objects into their string representations.*
- #define Colr_rjust(text, justwidth, ...)

  *Sets the JustifyMethod for a ColorText while allocating it.*
- #define Colr_rjust_char(text, justwidth, c, ...)

  *Sets the JustifyMethod for a ColorText while allocating it.*
- #define colr_snprintf(...) colr_printf_macro(snprintf, __VA_ARGS__)

  *Ensure colr_printf_register() has been called, and then call snprintf.*
- #define colr_sprintf(...) colr_printf_macro(sprintf, __VA_ARGS__)

  *Ensure colr_printf_register() has been called, and then call sprintf.*
- #define colr_str_either(s1, s2, s3) (colr_str_eq(s1, s2) || colr_str_eq(s1, s3))

  *Convenience macro for !strcmp(s1, s2) || !strcmp(s1, s3).*
- #define colr_str_eq(s1, s2)

  *Convenience macro for !strcmp(s1, s2).*
- #define colr_to_str(x)

  *Calls the <type>_to_str functions for the supported types.*
- #define COLR_VERSION "0.3.6"

  *Current version for ColrC.*
- #define Colra(text, ...) ColorText_from_values(text, __VA_ARGS__, _ColrLastArg)

  *Returns an initialized stack-allocated ColorText.*
- #define ColrResult(s) ColorResult_to_ptr(ColorResult_new(s))

  *Wraps an allocated string in a ColorResult, which marks it as "freeable" in the colr macros.*
- #define ext(x) ((ExtendedValue)x)

  *Casts to ExtendedValue (unsigned char).*
- #define ext_hex(s) ext_hex_or(s, ext(0))

  *Like hex(), but force a conversion to the closest ExtendedValue (256-colors).*
- #define ext_hex_or(s, default_value) ExtendedValue_from_hex_default(s, default_value)

  *Like hex_or(), but force a conversion to the closest ExtendedValue (256-colors).*
- #define EXT_INVALID COLOR_INVALID

  *Alias for COLOR_INVALID.*
- #define EXT_INVALID_RANGE COLOR_INVALID_RANGE

  *Possible error return value for ExtendedValue_from_str() or ExtendedValue_from_esc().*
- #define ext_rgb(r, g, b) ExtendedValue_from_RGB((RGB){.red=r, .green=g, .blue=b})

  *Creates the closest matching ExtendedValue from separate red, green, and blue values.*
- #define ext_RGB(rgbval) ExtendedValue_from_RGB(rgbval)

  *Creates the closest matching ExtendedValue from an RGB value.*
- #define fore(x) ColorArg_to_ptr(fore_arg(x))

  *Create a fore color suitable for use with the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros.*
- #define fore_arg(x)

  *Uses ColorArg_from_<type> to build a ColorArg with the appropriate color type, based on the type of it's argument.*
- #define fore_str(x) ColorArg_to_esc(fore_arg(x))

*Return just the escape code string for a fore color.*

- #define fore_str_static(x)

  *Creates a stack-allocated escape code string (`char*`) for a fore color.*
- #define hex(s) hex_or(s, rgb(0, 0, 0))

  *Use RGB_from_hex_default() to create an RGB value.*
- #define hex_or(s, default_rgb) RGB_from_hex_default(s, default_rgb)

  *Use RGB_from_hex_default() to create an RGB value.*
- #define if_not_asprintf(...) if (asprintf(__VA_ARGS__) < 1)

  *Convenience macro for checking asprintf's return value.*
- #define NC CODE_RESET_ALL

  *Short-hand for CODE_RESET_ALL, stands for "No Color".*
- #define NCNL CODE_RESET_ALL "\n"

  *Short-hand for `CODE_RESET_ALL "\n"`, stands for "No Color, New Line".*
- #define rgb(r, g, b) ((RGB){.red=r, .green=g, .blue=b})

  *Creates an anonymous RGB struct for use in function calls.*
- #define style(x) ColorArg_to_ptr(style_arg(x))

  *Create a style suitable for use with the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros.*
- #define style_arg(x)

  *Uses ColorArg_from_StyleValue to build a ColorArg with the appropriate color type/value.*
- #define STYLE_LEN 6

  *Maximum length for a style escape code, including "\0".*
- #define STYLE_LEN_MIN 5

  *Minimum length for the shortest style escape code, including "\0".*
- #define STYLE_MAX_VALUE ((StyleValue)OVERLINE)

  *Maximum value allowed for a StyleValue.*
- #define STYLE_MIN_VALUE ((StyleValue)STYLE_INVALID_RANGE)

  *Minimum value allowed for a StyleValue.*
- #define style_str(x) ColorArg_to_esc(style_arg(x))

  *Return just the escape code string for a style.*
- #define style_str_static(x)

  *A less-flexible style_str() that returns a static escape code string for a style.*
- #define WCODE_RESET_ALL L"\x1b[0m"

  *Convenience definition for wide chars.*
- #define WCODE_RESET_BACK L"\x1b[49m"

  *Convenience definition for resetting the back color.*
- #define WCODE_RESET_FORE L"\x1b[39m"

  *Convenience definition for resetting the fore color.*
- #define while_colr_va_arg(ap, vartype, x) while (x = va_arg(ap, vartype), !_colr_is_last_arg(x))

  *Construct a while-loop over a `va_list`, where the last argument is expected to be _ColrLastArg, or a pointer to a _ColrLastArg_s with the same values as _ColrLastArg.*
- #define WNC WCODE_RESET_ALL

  *Short-hand for WCODE_RESET_ALL, stands for "Wide No Color".*
- #define WNCNL WCODE_RESET_ALL L"\n"

  *Short-hand for `WCODE_RESET_ALL "\n"`, stands for "Wide No Color, New Line".*

## Typedefs

- typedef unsigned char ExtendedValue

  *Convenience `typedef` for clarity when dealing with extended (256) colors.*
- typedef char *(* rainbow_creator) (const char *s, double freq, size_t offset, size_t spread)

  *A function type that knows how to create rainbowized text.*
- typedef void(* RGB_fmter) (char *out, RGB rgb)

  *A function type that knows how to fill a string with an rgb escape code.*

Enumerations

- enum ArgType {
  **ARGTYPE_NONE** = -1,
  **FORE** = 0,
  **BACK** = 1,
  **STYLE** = 2 }

  *Argument types (fore, back, style).*
- enum BasicValue {
  **BASIC_INVALID_RANGE** = COLOR_INVALID_RANGE,
  **BASIC_INVALID** = COLOR_INVALID,
  **BASIC_NONE** = -1,
  **BLACK** = 0,
  **RED** = 1,
  **GREEN** = 2,
  **YELLOW** = 3,
  **BLUE** = 4,
  **MAGENTA** = 5,
  **CYAN** = 6,
  **WHITE** = 7,
  **UNUSED** = 8,
  **RESET** = 9,
  **LIGHTBLACK** = 10,
  **LIGHTRED** = 11,
  **LIGHTGREEN** = 12,
  **LIGHTYELLOW** = 13,
  **LIGHTBLUE** = 14,
  **LIGHTMAGENTA** = 15,
  **LIGHTCYAN** = 16,
  **LIGHTWHITE** = 17 }

  *Basic color values, with a few convenience values for extended colors.*
- enum ColorJustifyMethod {
  **JUST_NONE** = -1,
  **JUST_LEFT** = 0,
  **JUST_RIGHT** = 1,
  **JUST_CENTER** = 2 }

  *Justification style for ColorTexts.*
- enum ColorType {
  **TYPE_NONE** = -6,
  **TYPE_INVALID_STYLE** = -5,
  **TYPE_INVALID_RGB_RANGE** = -4,
  **TYPE_INVALID_EXT_RANGE** = COLOR_INVALID_RANGE,
  **TYPE_INVALID** = COLOR_INVALID,
  **TYPE_BASIC** = 0,
  **TYPE_EXTENDED** = 1,
  **TYPE_RGB** = 2,
  **TYPE_STYLE** = 3 }

  *Color/Style code types. Used with ColorType_from_str() and ColorValue.*
- enum StyleValue {
  **STYLE_INVALID_RANGE** = COLOR_INVALID_RANGE,
  **STYLE_INVALID** = COLOR_INVALID,
  **STYLE_NONE** = -1,
  **RESET_ALL** = 0,
  **BRIGHT** = 1,
  **DIM** = 2,
  **ITALIC** = 3,
  **UNDERLINE** = 4,

**FLASH** = 5,
**HIGHLIGHT** = 7,
**STRIKETHRU** = 9,
**NORMAL** = 22,
**FRAME** = 51,
**ENCIRCLE** = 52,
**OVERLINE** = 53 }

*Style values.*

## Functions

- void _colr_free (void *p)

  *Calls Colr *_free() functions for Colr objects, otherwise just calls free().*
- bool _colr_is_last_arg (void *p)

  *Determines if a void pointer is _ColrLastArg (the last-arg-marker).*
- char * _colr_join (void *joinerp,...)

  *Joins ColorArgs, ColorTexts, and strings (char*) into one long string separated by it's first argument.*
- size_t _colr_join_array_length (void *ps)

  *Determine the length of a NULL-terminated array of strings (char*), ColorArgs, ColorResults, or ColorTexts.*
- size_t _colr_join_arrayn_size (void *joinerp, void *ps, size_t count)

  *Get the size in bytes needed to join an array of strings (char*), ColorArgs, ColorResults, or ColorTexts by another string (char*), ColorArg, ColorResult, or ColorText.*
- size_t _colr_join_size (void *joinerp, va_list args)

  *Parse arguments, just as in _colr_join(), but only return the size needed to allocate the resulting string.*
- size_t _colr_ptr_length (void *p)

  *Get the size, in bytes, needed to convert a ColorArg, ColorResult, ColorText, or string (char*) into a string.*
- char * _colr_ptr_repr (void *p)

  *Determine what kind of pointer is being passed, and call the appropriate <type>_repr function to obtain an allocated string representation.*
- char * _colr_ptr_to_str (void *p)

  *Determine what kind of pointer is being passed, and call the appropriate <type>_to_str function to obtain an allocated string.*
- char * _rainbow (RGB_fmter fmter, const char *s, double freq, size_t offset, size_t spread)

  *Handles multibyte character string (char*) conversion and character iteration for all of the rainbow_functions.*
- bool ArgType_eq (ArgType a, ArgType b)

  *Compares two ArgTypes.*
- char * ArgType_repr (ArgType type)

  *Creates a string (char*) representation of a ArgType.*
- char * ArgType_to_str (ArgType type)

  *Creates a human-friendly string (char*) from an ArgType.*
- bool BasicValue_eq (BasicValue a, BasicValue b)

  *Compares two BasicValues.*
- BasicValue BasicValue_from_esc (const char *s)

  *Convert an escape-code string (char*) to an actual BasicValue enum value.*
- BasicValue BasicValue_from_str (const char *arg)

  *Convert named argument to an actual BasicValue enum value.*
- bool BasicValue_is_invalid (BasicValue bval)

  *Determines whether a BasicValue is invalid.*
- bool BasicValue_is_valid (BasicValue bval)

  *Determines whether a BasicValue is valid.*

- char * BasicValue_repr (BasicValue bval)

  *Creates a string (char*) representation of a BasicValue.*
- int BasicValue_to_ansi (ArgType type, BasicValue bval)

  *Converts a fore/back BasicValue to the actual ansi code number.*
- char * BasicValue_to_str (BasicValue bval)

  *Create a human-friendly string (char*) representation for a BasicValue.*
- ColorArg ColorArg_empty (void)

  *Create a ColorArg with ARGTYPE_NONE and ColorValue.type.TYPE_NONE.*
- bool ColorArg_eq (ColorArg a, ColorArg b)

  *Compares two ColorArg structs.*
- char * ColorArg_example (ColorArg carg, bool colorized)

  *Create a string (char*) representation of a ColorArg with a stylized type/name using escape codes built from the ColorArg's values.*
- void ColorArg_free (ColorArg *p)

  *Free allocated memory for a ColorArg.*
- ColorArg ColorArg_from_BasicValue (ArgType type, BasicValue value)

  *Explicit version of ColorArg_from_value that only handles BasicValues.*
- ColorArg ColorArg_from_esc (const char *s)

  *Parse an escape-code string (char*) into a ColorArg.*
- ColorArg ColorArg_from_ExtendedValue (ArgType type, ExtendedValue value)

  *Explicit version of ColorArg_from_value that only handles ExtendedValues.*
- ColorArg ColorArg_from_RGB (ArgType type, RGB value)

  *Explicit version of ColorArg_from_value that only handles RGB structs.*
- ColorArg ColorArg_from_str (ArgType type, const char *colorname)

  *Build a ColorArg (fore, back, or style value) from a known color name/style.*
- ColorArg ColorArg_from_StyleValue (ArgType type, StyleValue value)

  *Explicit version of ColorArg_from_value that only handles StyleValues.*
- ColorArg ColorArg_from_value (ArgType type, ColorType colrtype, void *p)

  *Used with the color_arg macro to dynamically create a ColorArg based on it's argument type.*
- bool ColorArg_is_empty (ColorArg carg)

  *Checks to see if a ColorArg is an empty placeholder.*
- bool ColorArg_is_invalid (ColorArg carg)

  *Checks to see if a ColorArg holds an invalid value.*
- bool ColorArg_is_ptr (void *p)

  *Checks a void pointer to see if it contains a ColorArg struct.*
- bool ColorArg_is_valid (ColorArg carg)

  *Checks to see if a ColorArg holds a valid value.*
- size_t ColorArg_length (ColorArg carg)

  *Returns the length in bytes needed to allocate a string (char*) built with ColorArg_to_esc().*
- char * ColorArg_repr (ColorArg carg)

  *Creates a string (char*) representation for a ColorArg.*
- char * ColorArg_to_esc (ColorArg carg)

  *Converts a ColorArg into an escape code string (char*).*
- bool ColorArg_to_esc_s (char *dest, ColorArg carg)

  *Converts a ColorArg into an escape code string (char*) and fills the destination string.*
- ColorArg * ColorArg_to_ptr (ColorArg carg)

  *Copies a ColorArg into memory and returns the pointer.*
- void ColorArgs_array_free (ColorArg **ps)

  *Free an allocated array of ColorArgs, including the array itself.*
- char * ColorArgs_array_repr (ColorArg **lst)

  *Creates a string representation for an array of ColorArg pointers.*

- ColorArg ** ColorArgs_from_str (const char *s, bool unique)

  *Create an array of ColorArgs from escape-codes found in a string (`char*`).*
- ColorJustify ColorJustify_empty (void)

  *Creates an "empty" ColorJustify, with JUST_NONE set.*
- bool ColorJustify_eq (ColorJustify a, ColorJustify b)

  *Compares two ColorJustify structs.*
- bool ColorJustify_is_empty (ColorJustify cjust)

  *Checks to see if a ColorJustify is "empty".*
- ColorJustify ColorJustify_new (ColorJustifyMethod method, int width, char padchar)

  *Creates a ColorJustify.*
- char * ColorJustify_repr (ColorJustify cjust)

  *Creates a string (`char*`) representation for a ColorJustify.*
- char * ColorJustifyMethod_repr (ColorJustifyMethod meth)

  *Creates a string (`char*`) representation for a ColorJustifyMethod.*
- ColorResult ColorResult_empty (void)

  *Creates a ColorResult with `.result=NULL` and `.length=-1`, with the appropriate struct marker.*
- bool ColorResult_eq (ColorResult a, ColorResult b)

  *Compares two ColorResults.*
- void ColorResult_free (ColorResult *p)

  *Free allocated memory for a ColorResult and it's `.result` member.*
- bool ColorResult_is_ptr (void *p)

  *Checks a void pointer to see if it contains a ColorResult struct.*
- size_t ColorResult_length (ColorResult cres)

  *Return the length in bytes (including the null-terminator), that is needed to store the return from ColorResult_to_str() (`.result`).*
- ColorResult ColorResult_new (char *s)

  *Initialize a new ColorResult with an allocated string (`char*`).*
- char * ColorResult_repr (ColorResult cres)

  *Create a string representation for a ColorResult.*
- ColorResult * ColorResult_to_ptr (ColorResult cres)

  *Allocate memory for a ColorResult, fill it, and return it.*
- char * ColorResult_to_str (ColorResult cres)

  *Convert a ColorResult into a string (`char*`).*
- ColorText ColorText_empty (void)

  *Creates an "empty" ColorText with pointers set to `NULL`.*
- void ColorText_free (ColorText *p)

  *Frees a ColorText and it's ColorArgs.*
- void ColorText_free_args (ColorText *p)

  *Frees the ColorArg members of a ColorText.*
- ColorText ColorText_from_values (char *text,...)

  *Builds a ColorText from 1 mandatory string (`char*`), and optional fore, back, and style args (pointers to ColorArgs).*
- bool ColorText_has_arg (ColorText ctext, ColorArg carg)

  *Checks to see if a ColorText has a certain ColorArg value set.*
- bool ColorText_has_args (ColorText ctext)

  *Checks to see if a ColorText has any argument values set.*
- bool ColorText_is_empty (ColorText ctext)

  *Checks to see if a ColorText has no usable values.*
- bool ColorText_is_ptr (void *p)

  *Checks a void pointer to see if it contains a ColorText struct.*
- size_t ColorText_length (ColorText ctext)

*Returns the length in bytes needed to allocate a string (char*) built with ColorText_to_str() with the current text, fore, back, and style members.*

- char * ColorText_repr (ColorText ctext)

  *Allocate a string (char*) representation for a ColorText.*
- ColorText * ColorText_set_just (ColorText *ctext, ColorJustify cjust)

  *Set the ColorJustify method for a ColorText, and return the ColorText.*
- void ColorText_set_values (ColorText *ctext, char *text,...)

  *Initializes an existing ColorText from 1 mandatory string (char*), and optional fore, back, and style args (pointers to ColorArgs).*
- ColorText * ColorText_to_ptr (ColorText ctext)

  *Copies a ColorText into allocated memory and returns the pointer.*
- char * ColorText_to_str (ColorText ctext)

  *Stringifies a ColorText struct, creating a mix of escape codes and text.*
- bool ColorType_eq (ColorType a, ColorType b)

  *Compares two ColorTypes.*
- ColorType ColorType_from_str (const char *arg)

  *Determine which type of color value is desired by name.*
- bool ColorType_is_invalid (ColorType type)

  *Check to see if a ColorType value is considered invalid.*
- bool ColorType_is_valid (ColorType type)

  *Check to see if a ColorType value is considered valid.*
- char * ColorType_repr (ColorType type)

  *Creates a string (char*) representation of a ColorType.*
- char * ColorType_to_str (ColorType type)

  *Create a human-friendly string (char*) representation for a ColorType.*
- ColorValue ColorValue_empty (void)

  *Create an "empty" ColorValue.*
- bool ColorValue_eq (ColorValue a, ColorValue b)

  *Compares two ColorValue structs.*
- char * ColorValue_example (ColorValue cval)

  *Create a string (char*) representation of a ColorValue with a human-friendly type/name.*
- ColorValue ColorValue_from_esc (const char *s)

  *Convert an escape-code string (char*) into a ColorValue.*
- ColorValue ColorValue_from_str (const char *s)

  *Create a ColorValue from a known color name, or RGB string (char*).*
- ColorValue ColorValue_from_value (ColorType type, void *p)

  *Used with the color_val macro to dynamically create a ColorValue based on it's argument type.*
- bool ColorValue_has_BasicValue (ColorValue cval, BasicValue bval)

  *Checks to see if a ColorValue has a BasicValue set.*
- bool ColorValue_has_ExtendedValue (ColorValue cval, ExtendedValue eval)

  *Checks to see if a ColorValue has a ExtendedValue set.*
- bool ColorValue_has_RGB (ColorValue cval, RGB rgb)

  *Checks to see if a ColorValue has a RGB value set.*
- bool ColorValue_has_StyleValue (ColorValue cval, StyleValue sval)

  *Checks to see if a ColorValue has a StyleValue set.*
- bool ColorValue_is_empty (ColorValue cval)

  *Checks to see if a ColorValue is an empty placeholder.*
- bool ColorValue_is_invalid (ColorValue cval)

  *Checks to see if a ColorValue holds an invalid value.*
- bool ColorValue_is_valid (ColorValue cval)

  *Checks to see if a ColorValue holds a valid value.*

- size_t ColorValue_length (ArgType type, ColorValue cval)

  *Returns the length in bytes needed to allocate a string (char*) built with ColorValue_to_esc() with the specified ArgType and ColorValue.*
- char * ColorValue_repr (ColorValue cval)

  *Creates a string (char*) representation of a ColorValue.*
- char * ColorValue_to_esc (ArgType type, ColorValue cval)

  *Converts a ColorValue into an escape code string (char*).*
- bool ColorValue_to_esc_s (char *dest, ArgType type, ColorValue cval)

  *Converts a ColorValue into an escape code string (char*) and fills the destination string.*
- regmatch_t * colr_alloc_regmatch (regmatch_t match)

  *Allocates space for a regmatch_t, initializes it, and returns a pointer to it.*
- void colr_append_reset (char *s)

  *Appends CODE_RESET_ALL to a string (char*), but makes sure to do it before any newlines.*
- char colr_char_escape_char (const char c)

  *Returns the char needed to represent an escape sequence in C.*
- bool colr_char_in_str (const char *s, const char c)

  *Determines if a character exists in the given string (char*).*
- bool colr_char_is_code_end (const char c)

  *Determines if a character is suitable for an escape code ending.*
- char * colr_char_repr (char c)

  *Creates a string (char*) representation for a char.*
- bool colr_char_should_escape (const char c)

  *Determines if an ascii character has an escape sequence in C.*
- bool colr_check_marker (uint32_t marker, void *p)

  *Checks an unsigned int against the individual bytes behind a pointer's value.*
- char * colr_empty_str (void)

  *Allocates an empty string (char*).*
- void colr_free_re_matches (regmatch_t **matches)

  *Free an array of allocated regmatch_t, like the return from colr_re_matches().*
- char * colr_join_array (void *joinerp, void *ps)

  *Join an array of strings (char*), ColorArgs, or ColorTexts by another string (char*), ColorArg, or ColorText.*
- char * colr_join_arrayn (void *joinerp, void *ps, size_t count)

  *Join an array of strings (char*), ColorArgs, or ColorTexts by another string (char*), ColorArg, or ColorText.*
- size_t colr_mb_len (const char *s, size_t length)

  *Like mbrlen, except it will return the length of the next N (length) multibyte characters in bytes.*
- int colr_printf_handler (FILE *fp, const struct printf_info *info, const void *const *args)

  *Handles printing with printf for Colr objects.*
- int colr_printf_info (const struct printf_info *info, size_t n, int *argtypes, int *sz)

  *Handles the arg count/size for the Colr printf handler.*
- void colr_printf_register (void)

  *Registers COLR_FMT_CHAR to handle Colr objects in the printf-family functions.*
- regmatch_t ** colr_re_matches (const char *s, regex_t *repattern)

  *Returns all regmatch_t matches for regex pattern in a string (char*).*
- bool colr_set_locale (void)

  *Sets the locale to (LC_ALL, "") if it hasn't already been set.*
- bool colr_str_array_contains (char **lst, const char *s)

  *Determine if a string (char*) is in an array of strings (char**, where the last element is NULL).*
- void colr_str_array_free (char **ps)

  *Free an allocated array of strings, including the array itself.*

- char * colr_str_center (const char *s, int width, const char padchar)

  *Center-justifies a string (char*), ignoring escape codes when measuring the width.*
- size_t colr_str_char_count (const char *s, const char c)

  *Counts the number of characters (c) that are found in a string (char*) (s).*
- size_t colr_str_char_lcount (const char *s, const char c)

  *Counts the number of characters (c) that are found at the beginning of a string (char*) (s).*
- size_t colr_str_chars_lcount (const char *restrict s, const char *restrict chars)

  *Counts the number of characters that are found at the beginning of a string (char*) (s), where the character can be any of chars.*
- size_t colr_str_code_count (const char *s)

  *Return the number of escape-codes in a string (char*).*
- size_t colr_str_code_len (const char *s)

  *Return the number of bytes that make up all the escape-codes in a string (char*).*
- char * colr_str_copy (char *restrict dest, const char *restrict src, size_t length)

  *Copies a string (char*) like strncpy, but ensures null-termination.*
- bool colr_str_ends_with (const char *restrict s, const char *restrict suffix)

  *Determine if one string (char*) ends with another.*
- char ** colr_str_get_codes (const char *s, bool unique)

  *Get an array of escape-codes from a string (char*).*
- bool colr_str_has_codes (const char *s)

  *Determines if a string (char*) has ANSI escape codes in it.*
- ColrHash colr_str_hash (const char *s)

  *Hash a string using djb2.*
- bool colr_str_is_all (const char *s, const char c)

  *Determines whether a string (char*) consists of only one character, possibly repeated.*
- bool colr_str_is_codes (const char *s)

  *Determines if a string (char*) is composed entirely of escape codes.*
- bool colr_str_is_digits (const char *s)

  *Determines whether all characters in a string (char*) are digits.*
- char * colr_str_ljust (const char *s, int width, const char padchar)

  *Left-justifies a string (char*), ignoring escape codes when measuring the width.*
- void colr_str_lower (char *s)

  *Converts a string (char*) into lower case in place.*
- size_t colr_str_lstrip (char *restrict dest, const char *restrict s, size_t length, const char c)

  *Strip a leading character from a string (char*), filling another string (char*) with the result.*
- char * colr_str_lstrip_char (const char *s, const char c)

  *Strips a leading character from a string (char*), and allocates a new string with the result.*
- char * colr_str_lstrip_chars (const char *restrict s, const char *restrict chars)

  *Removes certain characters from the start of a string (char*) and allocates a new string with the result.*
- size_t colr_str_mb_len (const char *s)

  *Returns the number of characters in a string (char*), taking into account possibly multibyte characters.*
- size_t colr_str_noncode_len (const char *s)

  *Returns the length of string (char*), ignoring escape codes and the the null-terminator.*
- char * colr_str_replace (const char *restrict s, const char *restrict target, const char *restrict repl)

  *Replaces the first substring found in a string (char*).*
- char * colr_str_replace_all (const char *restrict s, const char *restrict target, const char *restrict repl)

  *Replaces the first substring found in a string (char*).*
- char * colr_str_replace_all_ColorArg (const char *restrict s, const char *restrict target, ColorArg *repl)

*Replace all substrings in a string (char*) with a ColorArg's string result.*

- char * colr_str_replace_all_ColorResult (const char *restrict s, const char *restrict target, ColorResult *repl)

  *Replace all substrings in a string (char*) with a ColorResult's string result.*
- char * colr_str_replace_all_ColorText (const char *restrict s, const char *restrict target, ColorText *repl)

  *Replace all substrings in a string (char*) with a ColorText's string result.*
- char * colr_str_replace_cnt (const char *restrict s, const char *restrict target, const char *restrict repl, int count)

  *Replaces one or more substrings in a string (char*).*
- char * colr_str_replace_ColorArg (const char *restrict s, const char *restrict target, ColorArg *repl)

  *Replace a substring in a string (char*) with a ColorArg's string result.*
- char * colr_str_replace_ColorResult (const char *restrict s, const char *restrict target, ColorResult *repl)

  *Replace a substring in a string (char*) with a ColorResult's string result.*
- char * colr_str_replace_ColorText (const char *restrict s, const char *restrict target, ColorText *repl)

  *Replace a substring in a string (char*) with a ColorText's string result.*
- char * colr_str_replace_re (const char *restrict s, const char *restrict pattern, const char *restrict repl, int re_flags)

  *Replaces a substring from a regex pattern string (char*) in a string (char*).*
- char * colr_str_replace_re_all (const char *restrict s, const char *restrict pattern, const char *restrict repl, int re_flags)

  *Replaces all substrings from a regex pattern string (char*) in a string (char*).*
- char * colr_str_replace_re_all_ColorArg (const char *restrict s, const char *restrict pattern, ColorArg *repl, int re_flags)

  *Replace all substrings from a regex pattern string (char*) in a string (char*) with a ColorArg's string result.*
- char * colr_str_replace_re_all_ColorResult (const char *restrict s, const char *restrict pattern, ColorResult *repl, int re_flags)

  *Replace all substrings from a regex pattern string (char*) in a string (char*) with a ColorResult's string result.*
- char * colr_str_replace_re_all_ColorText (const char *restrict s, const char *restrict pattern, ColorText *repl, int re_flags)

  *Replace all substrings from a regex pattern string (char*) in a string (char*) with a ColorText's string result.*
- char * colr_str_replace_re_ColorArg (const char *restrict s, const char *restrict pattern, ColorArg *repl, int re_flags)

  *Replace substrings from a regex pattern string (char*) in a string (char*) with a ColorArg's string result.*
- char * colr_str_replace_re_ColorResult (const char *restrict s, const char *restrict pattern, ColorResult *repl, int re_flags)

  *Replace substrings from a regex pattern string (char*) in a string (char*) with a ColorResult's string result.*
- char * colr_str_replace_re_ColorText (const char *restrict s, const char *restrict pattern, ColorText *repl, int re_flags)

  *Replace substrings from a regex pattern string (char*) in a string (char*) with a ColorText's string result.*
- char * colr_str_replace_re_match (const char *restrict s, regmatch_t *match, const char *restrict repl)

  *Replaces substrings from a single regex match (regmatch_t*) in a string (char*).*
- char * colr_str_replace_re_match_ColorArg (const char *restrict s, regmatch_t *match, ColorArg *repl)

  *Replace substrings from a regex match (regmatch_t*) in a string (char*) with a ColorArg's string result.*

- char * colr_str_replace_re_match_ColorResult (const char *restrict s, regmatch_t *match, ColorResult *repl)

  *Replace substrings from a regex match (`regmatch_t*`) in a string (`char*`) with a ColorResult's string result.*
- char * colr_str_replace_re_match_ColorText (const char *restrict s, regmatch_t *match, ColorText *repl)

  *Replace substrings from a regex match (`regmatch_t*`) in a string (`char*`) with a ColorText's string result.*
- char * colr_str_replace_re_match_i (const char *restrict ref, char *target, regmatch_t *match, const char *restrict repl)

  *Replaces substrings from a regex match (`regmatch_t*`) in a string (`char*`).*
- char * colr_str_replace_re_matches (const char *restrict s, regmatch_t **matches, const char *restrict repl)

  *Replaces substrings from an array of regex match (`regmatch_t*`) in a string (`char*`).*
- char * colr_str_replace_re_matches_ColorArg (const char *restrict s, regmatch_t **matches, ColorArg *repl)

  *Replace substrings from an array of regex matches (`regmatch_t**`) in a string (`char*`) with a ColorArg's string result.*
- char * colr_str_replace_re_matches_ColorResult (const char *restrict s, regmatch_t **matches, ColorResult *repl)

  *Replace substrings from an array of regex matches (`regmatch_t**`) in a string (`char*`) with a ColorResult's string result.*
- char * colr_str_replace_re_matches_ColorText (const char *restrict s, regmatch_t **matches, ColorText *repl)

  *Replace substrings from an array of regex matches (`regmatch_t**`) in a string (`char*`) with a ColorText's string result.*
- char * colr_str_replace_re_pat (const char *restrict s, regex_t *repattern, const char *restrict repl)

  *Replaces regex patterns in a string (`char*`).*
- char * colr_str_replace_re_pat_all (const char *restrict s, regex_t *repattern, const char *restrict repl)

  *Replaces all matches to a regex pattern in a string (`char*`).*
- char * colr_str_replace_re_pat_all_ColorArg (const char *restrict s, regex_t *repattern, ColorArg *repl)

  *Replace all matches to a regex pattern in a string (`char*`) with a ColorArg's string result.*
- char * colr_str_replace_re_pat_all_ColorResult (const char *restrict s, regex_t *repattern, ColorResult *repl)

  *Replace all matches to a regex pattern in a string (`char*`) with a ColorResult's string result.*
- char * colr_str_replace_re_pat_all_ColorText (const char *restrict s, regex_t *repattern, ColorText *repl)

  *Replace all matches to a regex pattern in a string (`char*`) with a ColorText's string result.*
- char * colr_str_replace_re_pat_ColorArg (const char *restrict s, regex_t *repattern, ColorArg *repl)

  *Replace regex patterns in a string (`char*`) with a ColorArg's string result.*
- char * colr_str_replace_re_pat_ColorResult (const char *restrict s, regex_t *repattern, ColorResult *repl)

  *Replace regex patterns in a string (`char*`) with a ColorResult's string result.*
- char * colr_str_replace_re_pat_ColorText (const char *restrict s, regex_t *repattern, ColorText *repl)

  *Replace regex patterns in a string (`char*`) with a ColorText's string result.*
- char * colr_str_repr (const char *s)

  *Convert a string (`char*`) into a representation of a string, by wrapping it in quotes and escaping characters that need escaping.*
- char * colr_str_rjust (const char *s, int width, const char padchar)

  *Right-justifies a string (`char*`), ignoring escape codes when measuring the width.*

- bool colr_str_starts_with (const char *restrict s, const char *restrict prefix)

  *Checks a string (`char*`) for a certain prefix substring.*
- char * colr_str_strip_codes (const char *s)

  *Strips escape codes from a string (`char*`), resulting in a new allocated string.*
- char * colr_str_to_lower (const char *s)

  *Allocate a new lowercase version of a string (`char*`).*
- bool colr_supports_rgb (void)

  *Determine whether the current environment support RGB (True Colors).*
- bool colr_supports_rgb_static (void)

  *Same as colr_supports_rgb(), but the environment is only checked on the first call.*
- TermSize colr_term_size (void)

  *Attempts to retrieve the row/column size of the terminal and returns a TermSize.*
- struct winsize colr_win_size (void)

  *Attempts to retrieve a `winsize` struct from an `ioctl` call.*
- struct winsize colr_win_size_env (void)

  *Get window/terminal size using the environment variables `LINES`, `COLUMNS`, or `COLS`.*
- bool ExtendedValue_eq (ExtendedValue a, ExtendedValue b)

  *Compares two ExtendedValues.*
- int ExtendedValue_from_BasicValue (BasicValue bval)

  *Convert a BasicValue into an ExtendedValue.*
- int ExtendedValue_from_esc (const char *s)

  *Convert an escape-code string (`char*`) to an ExtendedValue.*
- int ExtendedValue_from_hex (const char *hexstr)

  *Create an ExtendedValue from a hex string (`char*`).*
- ExtendedValue ExtendedValue_from_hex_default (const char *hexstr, ExtendedValue default_value)

  *Create an ExtendedValue from a hex string (`char*`), but return a default value if the hex string is invalid.*
- ExtendedValue ExtendedValue_from_RGB (RGB rgb)

  *Convert an RGB value into the closest matching ExtendedValue.*
- int ExtendedValue_from_str (const char *arg)

  *Converts a known name, integer string (0-255), or a hex string (`char*`), into an ExtendedValue suitable for the extended-value-based functions.*
- bool ExtendedValue_is_invalid (int eval)

  *Determines whether an integer is an invalid ExtendedValue.*
- bool ExtendedValue_is_valid (int eval)

  *Determines whether an integer is a valid ExtendedValue.*
- char * ExtendedValue_repr (int eval)

  *Creates a string (`char*`) representation of a ExtendedValue.*
- char * ExtendedValue_to_str (ExtendedValue eval)

  *Creates a human-friendly string (`char*`) from an ExtendedValue's actual value, suitable for use with ExtendedValue_from_str().*
- void format_bg (char *out, BasicValue value)

  *Create an escape code for a background color.*
- void format_bg_RGB (char *out, RGB rgb)

  *Create an escape code for a true color (rgb) background color using values from an RGB struct.*
- void format_bg_RGB_term (char *out, RGB rgb)

  *Create an escape code for a true color (rgb) fore color using an RGB struct's values, approximating 256-color values.*
- void format_bgx (char *out, unsigned char num)

  *Create an escape code for an extended background color.*
- void format_fg (char *out, BasicValue value)

*Create an escape code for a fore color.*

- void format_fg_RGB (char *out, RGB rgb)

  *Create an escape code for a true color (rgb) fore color using an RGB struct's values.*
- void format_fg_RGB_term (char *out, RGB rgb)

  *Create an escape code for a true color (rgb) fore color using an RGB struct's values, approximating 256-color values.*
- void format_fgx (char *out, unsigned char num)

  *Create an escape code for an extended fore color.*
- void format_style (char *out, StyleValue style)

  *Create an escape code for a style.*
- char * rainbow_bg (const char *s, double freq, size_t offset, size_t spread)

  *Rainbow-ize some text using rgb back colors, lolcat style.*
- char * rainbow_bg_term (const char *s, double freq, size_t offset, size_t spread)

  *This is exactly like rainbow_bg(), except it uses colors that are closer to the standard 256-color values.*
- char * rainbow_fg (const char *s, double freq, size_t offset, size_t spread)

  *Rainbow-ize some text using rgb fore colors, lolcat style.*
- char * rainbow_fg_term (const char *s, double freq, size_t offset, size_t spread)

  *This is exactly like rainbow_fg(), except it uses colors that are closer to the standard 256-color values.*
- RGB rainbow_step (double freq, size_t offset)

  *A single step in rainbow-izing produces the next color in the "rainbow" as an RGB value.*
- unsigned char RGB_average (RGB rgb)

  *Return the average for an RGB value.*
- bool RGB_eq (RGB a, RGB b)

  *Compare two RGB structs.*
- RGB RGB_from_BasicValue (BasicValue bval)

  *Return an RGB value from a known BasicValue.*
- int RGB_from_esc (const char *s, RGB *rgb)

  *Convert an escape-code string (`char*`) to an actual RGB value.*
- RGB RGB_from_ExtendedValue (ExtendedValue eval)

  *Return an RGB value from a known ExtendedValue.*
- int RGB_from_hex (const char *hexstr, RGB *rgb)

  *Convert a hex color into an RGB value.*
- RGB RGB_from_hex_default (const char *hexstr, RGB default_value)

  *Convert a hex color into an RGB value, but use a default value when errors occur.*
- int RGB_from_str (const char *arg, RGB *rgb)

  *Convert an RGB string (`char*`) into an RGB value.*
- RGB RGB_grayscale (RGB rgb)

  *Return a grayscale version of an RGB value.*
- RGB RGB_inverted (RGB rgb)

  *Make a copy of an RGB value, with the colors "inverted" (like highlighting text in the terminal).*
- RGB RGB_monochrome (RGB rgb)

  *Convert an RGB value into either black or white, depending on it's average grayscale value.*
- char * RGB_repr (RGB rgb)

  *Creates a string (`char*`) representation for an RGB value.*
- char * RGB_to_hex (RGB rgb)

  *Converts an RGB value into a hex string (`char*`).*
- char * RGB_to_str (RGB rgb)

  *Convert an RGB value into a human-friendly RGB string (`char*`) suitable for input to RGB_from_str().*
- RGB RGB_to_term_RGB (RGB rgb)

  *Convert an RGB value into it's nearest terminal-friendly RGB value.*
- bool StyleValue_eq (StyleValue a, StyleValue b)

*Compares two StyleValues.*

- StyleValue StyleValue_from_esc (const char *s)

  *Convert an escape-code string (char*) to an actual StyleValue enum value.*
- StyleValue StyleValue_from_str (const char *arg)

  *Convert a named argument to actual StyleValue enum value.*
- bool StyleValue_is_invalid (StyleValue sval)

  *Determines whether a StyleValue is invalid.*
- bool StyleValue_is_valid (StyleValue sval)

  *Determines whether a StyleValue is valid.*
- char * StyleValue_repr (StyleValue sval)

  *Creates a string (char*) representation of a StyleValue.*
- char * StyleValue_to_str (StyleValue sval)

  *Create a human-friendly string (char*) representation for a StyleValue.*
- char * TermSize_repr (TermSize ts)

  *Create a string (char*) representation for a TermSize.*

Variables

- int colr_printf_esc_mod

  *Integer to test for the presence of the "escaped output modifier" in colr_printf_handler.*

#### 0.6.2.1 Detailed Description

Declarations for ColrC functions, enums, structs, etc.

Common macros and definitions are found here in colr.h, however the functions are documented in colr.c.

#### 0.6.2.2 Data Structure Documentation

##### 0.6.2.2.1 struct BasicInfo

Holds a known color name and it's `BasicValue`.

This is used for the `basic_names` array in colr.c.

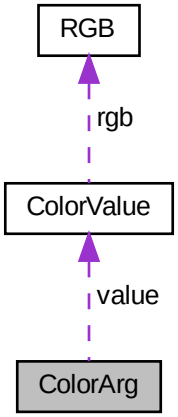
Data Fields

| char * | name | |
|---|---|---|
| BasicValue | value | |

##### 0.6.2.2.2 struct ColorArg

Holds an ArgType, and a ColorValue.

Collaboration diagram for ColorArg:




Data Fields

| | | |
|---|---|---|
| uint32_t | marker | A marker used to inspect void pointers and determine if they are ColorArgs. |
| ArgType | type | Fore, back, style, invalid. |
| ColorValue | value | Color type and value. |

#### 0.6.2.2.3 struct ColorJustify

Holds a string justification method, width, and padding character for ColorTexts.

Data Fields

| | | |
|---|---|---|
| uint32_t | marker | A marker used to inspect void pointers and determine if they are ColorJustifys. |
| ColorJustifyMethod | method | The justification method, can be JUST_NONE. |
| char | padchar | The desired padding character, or `0` to use the default (‘ ’). |
| int | width | The desired width for the final string, or `0` to use colr_term_size(). |

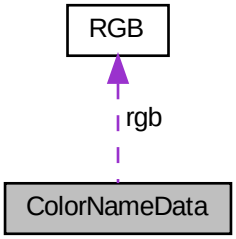
#### 0.6.2.2.4 struct ColorNameData

Holds info about a known color name, like it's ExtendedValue and it's RGB value.

Some of the names have the same ExtendedValue, and not all ExtendedValues have names.

This is used in the colr_name_data array.

Collaboration diagram for ColorNameData:




Data Fields

| | | |
|---|---|---|
| ExtendedValue | ext | ExtendedValue (256-colors) for the color. |
| char * | name | The known name of the color. |
| RGB | rgb | RGB (TrueColor) for the color. |

#### 0.6.2.2.5 struct ColorResult

Holds a string (`char*`) that was definitely allocated by Colr.

Examples:

ColorResult_example.c.

Data Fields

| | | |
|---|---|---|
| size_t | length | A length in bytes for the string result. Set when the ColorResult is initialized with a string (ColorResult_new()). Initially set to `-1`. |
| uint32_t | marker | A marker used to inspect void pointers and determine if they are ColorResults. |
| char * | result | A string (`char*`) result from one of the colr functions. |

#### 0.6.2.2.6 union ColorStructMarker

Breaks down Colr struct markers, such as COLORARG_MARKER, into individual bytes.

Data Fields

| | | |
|---|---|---|
| struct ColorStructMarker | bytes | Individual bytes that make up the marker. |
| uint32_t | marker | The actual uint32_t marker value. |

#### 0.6.2.2.7 struct ColorStructMarker.bytes

Individual bytes that make up the marker.

Data Fields

| | | |
|---|---|---|
| uint8_t | b1 | |
| uint8_t | b2 | |
| uint8_t | b3 | |
| uint8_t | b4 | |

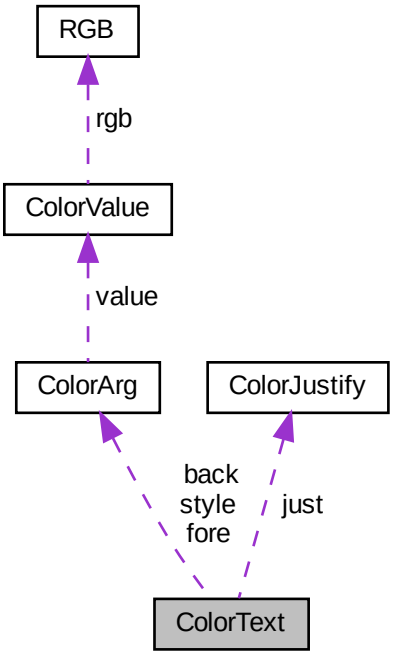
#### 0.6.2.2.8 struct ColorText

Holds a string of text, and optional fore, back, and style ColorArgs.

Examples:

colr_join_example.c, and simple_example.c.

Collaboration diagram for ColorText:




Data Fields

| | | |
|---|---|---|
| ColorArg * | back | ColorArg for back color. Can be NULL. |

Data Fields

| | | |
|---|---|---|
| ColorArg * | fore | ColorArg for fore color. Can be NULL. |
| ColorJustify | just | ColorJustify info, set to JUST_NONE by default. |
| uint32_t | marker | A marker used to inspect void pointers and determine if they are ColorTexts. |
| ColorArg * | style | ColorArg for style value. Can be NULL. |
| char * | text | Text to colorize. |

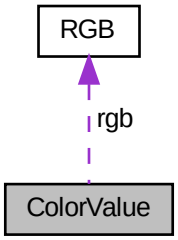
#### 0.6.2.2.9 struct ColorValue

Holds a color type and it's value.

The `.type` member must always match the type of color value it is holding.

This is internal. It's used to make the final interface easier to use. You probably shouldn't be using it.

Collaboration diagram for ColorValue:




Data Fields

| | | |
|---|---|---|
| BasicValue | basic | |
| ExtendedValue | ext | |
| RGB | rgb | |
| StyleValue | style | |
| ColorType | type | |

#### 0.6.2.2.10 struct ExtendedInfo

Holds a known color name and it's `ExtendedValue`.

This is used for the `basic_names` array in colr.c.

Data Fields

| | | |
|---|---|---|
| char * | name | |
| ExtendedValue | value | |

#### 0.6.2.2.11 struct RGB

Container for RGB values.

Data Fields

| | | |
|---|---|---|
| unsigned char | blue | Blue value for a color. |
| unsigned char | green | Green value for a color. |
| unsigned char | red | Red value for a color. |

#### 0.6.2.2.12 struct StyleInfo

Holds a known style name and it's `StyleValue`.

This is used for the `style_names` array in colr.c.

Data Fields

| | | |
|---|---|---|
| char * | name | |
| StyleValue | value | |

#### 0.6.2.2.13 struct TermSize

Holds a terminal size, usually retrieved with colr_term_size().

Data Fields

| | | |
|---|---|---|
| unsigned short | columns | |
| unsigned short | rows | |

### 0.6.2.3 Macro Definition Documentation

#### 0.6.2.3.1 alloc_basic

```
#define alloc_basic( ) calloc(CODE_LEN, sizeof(char))
```

Allocate enough for a basic code.

Returns

Pointer to the allocated string, or NULL on error.
*You must `free()` the memory allocated by this function.*

#### 0.6.2.3.2 alloc_extended

```
#define alloc_extended( ) calloc(CODEX_LEN, sizeof(char))
```

Allocate enough for an extended code.

Returns

Pointer to the allocated string, or NULL on error.
*You must `free()` the memory allocated by this function.*

#### 0.6.2.3.3 alloc_rgb

```
#define alloc_rgb( ) calloc(CODE_RGB_LEN, sizeof(char))
```

Allocate enough for an rgb code.

Returns

Pointer to the allocated string, or NULL on error.
*You must `free()` the memory allocated by this function.*

#### 0.6.2.3.4 alloc_style

```
#define alloc_style( ) calloc(STYLE_LEN, sizeof(char))
```

Allocate enough for a style code.

Returns

Pointer to the allocated string, or NULL on error.
*You must `free()` the memory allocated by this function.*

#### 0.6.2.3.5 asprintf_or_return

```
#define asprintf_or_return(
              retval,
              ...  ) if_not_asprintf(__VA_ARGS__) return retval
```

Convenience macro for bailing out of a function when asprintf fails.

Parameters

| | | |
|---|---|---|
| in | *retval* | Value to return if the asprintf fails. |
| in | ... | Arguments for asprintf. |

Referenced by BasicValue_to_str(), ColorArg_repr(), ColorArgs_array_repr(), ColorJustify_repr(), ColorText_repr(), colr_char_repr(), colr_str_center(), colr_str_ljust(), colr_str_replace_re_match(), colr_str_repr(), colr_str_rjust(), ExtendedValue_repr(), ExtendedValue_to_str(), RGB_repr(), RGB_↩ to_hex(), RGB_to_str(), StyleValue_to_str(), and TermSize_repr().

#### 0.6.2.3.6 back

```
#define back(
             x ) ColorArg_to_ptr(back_arg(x))
```

Create a back color suitable for use with the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros.

Technically, this macro accepts BasicValues, ExtendedValues, or RGB structs. However, for some of these you should be using the macros that create those things.

BasicValues can be used by their names (RED, YELLOW, etc.).

ExtendedValues can be created on the fly with ext().

RGB structs can be easily created with rgb().

Color names (`char*`) can be passed to generate the appropriate color value.

Parameters

| | | |
|---|---|---|
| in | *x* | A BasicValue, ExtendedValue, or RGB struct to use for the color value. |

Returns

A pointer to a heap-allocated ColorArg struct.
If used inside of the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros, *they will `free()` the result*. Otherwise, *you are responsible for calling `free()`*.

See also

back_arg
back_str
colr
Colr

Examples:

back_example.c, ColorResult_example.c, Colr_example.c, fore_example.c, and simple_↩ example.c.

#### 0.6.2.3.7 back_arg

```
#define back_arg(
              x )
```

**Value:**

```
_Generic( \
        (x), \
        char* : ColorArg_from_str, \
        RGB: ColorArg_from_RGB, \
        BasicValue: ColorArg_from_BasicValue, \
        ExtendedValue: ColorArg_from_ExtendedValue, \
        StyleValue: ColorArg_from_StyleValue \
    )(BACK, x)
```

Uses ColorArg_from_<type> to build a ColorArg with the appropriate color type, based on the type of it's argument.

Uses `_Generic` (C11 standard) to dynamically create a ColorArg. This is used by the back() macro.

Parameters

| | | |
|---|---|---|
| in | *x* | `BasicValue`, `Extended` (`unsigned char`), RGB struct, or string (color name) for back color. |

Returns

A ColorArg with the BACK type set, and it's `.value.type` set for the appropriate color type/value.
For invalid values the `.value.type` may be set to TYPE_INVALID.
*You must `free()` the memory allocated by this function.*

See also

back
back_str

#### 0.6.2.3.8 back_str

```
#define back_str(
              x ) ColorArg_to_esc(back_arg(x))
```

Return just the escape code string for a back color.

Parameters

| | | |
|---|---|---|
| in | *x* | A BasicValue, ExtendedValue, or RGB struct. |

Returns

An allocated string.
*You must `free()` the memory allocated by this function.*

See also

back
back_arg

#### 0.6.2.3.9 back_str_static

```
#define back_str_static(
            x )
```

**Value:**

```
__extension__ ({ \
        __typeof(x) _bss_val = x; \
        ColorArg _bss_carg = back_arg(_bss_val); \
        size_t _bss_len = ColorArg_length(_bss_carg); \
        char* _bss_codes = alloca(_bss_len); \
        ColorArg_to_esc_s(_bss_codes, _bss_carg); \
        _bss_codes; \
    })
```

Creates a stack-allocated escape code string (`char*`) for a back color.

These are not constant strings, but they are stored on the stack. A `Statement Expression` is used to build a string of the correct length and content using ColorArg_to_esc_s().

Attention

This feature uses a GNU extension, and is only available when COLR_GNU is defined. See the documentation for COLR_GNU.

Warning

This uses `alloca` to reserve space on the stack inside of a `Statement Expression`. A `Variable Length Array` will not work inside of a statement expression. If the call causes a stack overflow, program behavior is undefined. See previous links, and `here`.

You can also create stack-allocated escape code strings using format_bg(), format_bgx(), format_bg_RGB(), and format_bg_RGB_term().

Parameters

| | | |
|---|---|---|
| `in` | *x* | A BasicValue, ExtendedValue, or RGB value. |

Returns

A stack-allocated escape code string.

See also

back_str_static
style_str_static
format_fg
format_bg

#### 0.6.2.3.10 basic

```
#define basic(
              x ) ((BasicValue)(x))
```

Casts to BasicValue.

Parameters

| | | |
|---|---|---|
| in | *x* | Value to case to `BasicValue`. |

Returns

A BasicValue.

See also

fore
back
colr
Colr

#### 0.6.2.3.11 bool_colr_enum

```
#define bool_colr_enum(
              x ) (x < 0 ?  false:  true)
```

Returns the "truthiness" of the enums used in ColrC (BasicValue, ExtendedValue function-returns, StyleValue, ColorType, ArgType).

Any value less than `0` is considered false.

Parameters

| | | |
|---|---|---|
| in | *x* | An enum to convert to boolean. |

Return values

| | |
|---|---|
| *true* | if the value is considered valid, or non-empty. |
| *false* | if the value is considered invalid, or empty. |

Referenced by ColorArg_is_invalid(), ColorArg_is_valid(), ColorType_is_invalid(), ColorType_is_↩ valid(), ColorValue_is_invalid(), and ColorValue_is_valid().

#### 0.6.2.3.12 CODE_ANY_LEN

```
#define CODE_ANY_LEN 46
```

Maximum length in chars for any possible escape code mixture for one complete style (one of each: fore, back, and style).

(basically `(CODE_RGB_LEN * 2) + STYLE_LEN` since rgb codes are the longest).

Examples:

colr_printf_example.c.

#### 0.6.2.3.13 CODE_LEN

```
#define CODE_LEN 14
```

Maximum length for a basic fore/back escape code, including "\0".

Keep in mind that BasicValue actually has some "light" colors (104).

Referenced by format_bg(), and format_fg().

#### 0.6.2.3.14 CODE_LEN_MIN

```
#define CODE_LEN_MIN 5
```

Minimum length for the shortest basic fore/back escape code, including "\0".

Use CODE_LEN for allocation.

#### 0.6.2.3.15 CODE_RGB_LEN_MIN

```
#define CODE_RGB_LEN_MIN 14
```

Minimum length for the shortest RGB fore/back escape code, including "\0".

Use CODE_RGB_LEN for allocation.

#### 0.6.2.3.16 CODEX_LEN_MIN

```
#define CODEX_LEN_MIN 10
```

Minimum length for the shortest extended fore/back escape code, including "\0".

Use CODEX_LEN for allocation.

#### 0.6.2.3.17 color_arg

```
#define color_arg(
              type,
              x )
```

**Value:**

```
_Generic( \
        (x), \
        char*: ColorArg_from_str(type, x), \
        BasicValue: ColorArg_from_value(type, TYPE_BASIC, &x), \
        ExtendedValue: ColorArg_from_value(type, TYPE_EXTENDED, &x), \
        StyleValue: ColorArg_from_value(type, TYPE_STYLE, &x), \
        RGB: ColorArg_from_value(type, TYPE_RGB, &x) \
    )
```

Builds a correct ColorArg struct according to the type of it's second argument.

Uses `_Generic` (C11 standard) to dynamically create a ColorArg.

Parameters

| | | |
|---|---|---|
| in | *type* | ArgType (FORE, BACK, STYLE) to build the ColorArg. |
| in | *x* | BasicValue, Extended (unsigned char). or RGB value. |

Returns

ColorArg_from_value(type, [appropriate type], x)

#### 0.6.2.3.18 COLOR_LEN

```
#define COLOR_LEN 30
```

Maximum length in chars for any combination of basic/extended escape codes for one complete style (one of each: fore, back, style).

Should be `(CODEX_LEN * 2) + STYLE_LEN`. Allocating for a string that will be colorized must account for this.

#### 0.6.2.3.19 color_name_is_invalid

```
#define color_name_is_invalid(
              x ) ColorType_is_invalid(ColorType_from_str(x))
```

Convenience macro for checking if a color name is invalid.

Parameters

| | | |
|---|---|---|
| in | *x* | string (char*) to check (a name, hex-string, rgb-string, or integer-string). |

Returns

`true` if the name is an invalid color name, otherwise `false`.

See also

color_name_is_valid

#### 0.6.2.3.20 color_name_is_valid

```
#define color_name_is_valid(
            x ) ColorType_is_valid(ColorType_from_str(x))
```

Convenience macro for checking if a color name is valid.

Parameters

| | | |
|---|---|---|
| in | *x* | string (char*) to check (a name, hex-string, rgb-string, or integer-string). |

Returns

`true` if the name is a valid color name, otherwise `false`.

See also

color_name_is_invalid

#### 0.6.2.3.21 COLOR_RGB_LEN

```
#define COLOR_RGB_LEN 26
```

Maximum length in chars added to a rgb colorized string.

Should be `CODE_RGB_LEN + STYLE_LEN` Allocating for a string that will be colorized with rgb values must account for this.

#### 0.6.2.3.22 color_val

```
#define color_val(
              x )
```

**Value:**

```
_Generic( \
        (x), \
        BasicValue: ColorValue_from_value(TYPE_BASIC, &x), \
        ExtendedValue: ColorValue_from_value(TYPE_EXTENDED, &x), \
        StyleValue: ColorValue_from_value(TYPE_STYLE, &x), \
        RGB: ColorValue_from_value(TYPE_RGB, &x) \
    )
```

Builds a correct ColorValue struct according to the type of it's first argument.

Uses `_Generic` (C11 standard) to dynamically create a ColorValue.

Parameters

| | | |
|---|---|---|
| in | *x* | `BasicValue`, `Extended` (`unsigned char`). or RGB value. |

Returns

ColorValue_from_value([appropriate type], x)

#### 0.6.2.3.23 COLORARG_MARKER

```
#define COLORARG_MARKER UINT32_MAX
```

Marker for the ColorArg struct, for identifying a void pointer as a ColorArg.

Referenced by ColorArg_empty(), ColorArg_from_BasicValue(), ColorArg_from_esc(), ColorArg_↩ from_ExtendedValue(), ColorArg_from_RGB(), ColorArg_from_StyleValue(), ColorArg_from_value(), ColorArg_is_ptr(), and ColorArg_to_ptr().

#### 0.6.2.3.24 ColorValue_has

```
#define ColorValue_has(
              cval,
              val )
```

**Value:**

```
_Generic( \
        (val), \
        BasicValue: ColorValue_has_BasicValue, \
        ExtendedValue: ColorValue_has_ExtendedValue, \
        StyleValue: ColorValue_has_StyleValue, \
        RGB: ColorValue_has_RGB \
    )(cval, val)
```

Call the current ColorValue_has_<type> function for the given value.

Given the correct type of value, this will check to see if a ColorValue has the correct `.type` set for the value, and the values match.

Parameters

| | | |
|---|---|---|
| in | *cval* | The ColorValue to check. |
| in | *val* | A BasicValue, ExtendedValue, StyleValue, or RGB value. |

Returns

`true` if the ColorValue has the correct `.type` and it's value matches `val`, otherwise `false`.

See also

ColorValue
ColorValue_has_BasicValue
ColorValue_has_ExtendedValue
ColorValue_has_StyleValue
ColorValue_has_RGB

#### 0.6.2.3.25 Colr

```
#define Colr(
              text,
              ...  ) ColorText_to_ptr(ColorText_from_values(text, __VA_ARGS__, _ColrLastArg))
```

Returns a heap-allocated ColorText struct that can be used by itself, or with the colr_cat(), colr_↩ join(), Colr_cat(), and Colr_join() macros.

You must `free()` the resulting ColorText struct using ColorText_free(), unless you pass it to colr↩ _cat(), which will `free()` it for you.

Parameters

| | | |
|---|---|---|
| in | *text* | String to colorize/style. |
| in | ... | No more than 3 ColorArg pointers for fore, back, and style in any order. |

Returns

An allocated ColorText.
If used inside of the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros, *they will `free()` the result*. Otherwise, *you are responsible for calling `free()`*.

See also

Colra

Examples:

back_example.c, ColorResult_example.c, colr_cat_example.c, Colr_example.c, colr_join_↩ example.c, colr_printf_example.c, colr_replace_all_example.c, colr_replace_example.c, colr↩ _replace_re_all_example.c, colr_replace_re_example.c, fore_example.c, simple_example.c, and style_example.c.

#### 0.6.2.3.26 colr

```
#define colr(
             text,
             ... ) colr_cat(Colr(text, __VA_ARGS__))
```

Create an allocated string directly from Colr() arguments.

This is a wrapper around `colr_cat(Colr(text, ...))`, which will automatically `free()` the ColorText, and return a string that you are responsible for.

Parameters

| | | |
|---|---|---|
| in | *text* | String to colorize/style. |
| in | ... | No more than 3 ColorArg pointers for fore, back, and style in any order.<br>*colr_free() will be called on any ColorArg, ColorResult, or ColorText pointer passed to this function*. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

Examples:

ColorResult_example.c, Colr_example.c, and simple_example.c.

#### 0.6.2.3.27 colr_alloc_len

```
#define colr_alloc_len(
             x )
```

**Value:**

```
_Generic( \
        (x), \
        RGB: CODE_RGB_LEN, \
        BasicValue: CODE_LEN, \
        ExtendedValue: CODEX_LEN, \
        StyleValue: STYLE_LEN \
    )
```

Return the number of bytes needed to allocate an escape code string based on the color type.

Parameters

| | | |
|---|---|---|
| in | *x* | A BasicValue, ExtendedValue, RGB value, or StyleValue. |

Returns

The number of bytes needed to allocate a string using the color value.

#### 0.6.2.3.28 colr_asprintf

```
#define colr_asprintf(
              ...  ) colr_printf_macro(asprintf, __VA_ARGS__)
```

Ensure colr_printf_register() has been called, and then call `asprintf`.

Will call `free()` on any ColorArg pointer, ColorResult pointer, ColorText pointer, or the strings created by them.

Attention

This feature uses a GNU extension, and is only available when COLR_GNU is defined. See the documentation for COLR_GNU.

Parameters

| | | |
|---|---|---|
| in | ... | Arguments for 'asprintf<br>*colr_free() will be called on any ColorArg, ColorResult, or ColorText pointer passed to this function*. |

Returns

Same as `asprintf`.

Examples:

colr_printf_example.c.

#### 0.6.2.3.29 Colr_cat

```
#define Colr_cat(
              ...  ) ColorResult_to_ptr(ColorResult_new(colr_cat(__VA_ARGS__)))
```

Like colr_cat(), but returns an allocated ColorResult that the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros will automatically `free()`.

Parameters

| | | |
|---|---|---|
| in | ... | Arguments for colr_cat(), to concatenate.<br>*colr_free() will be called on any ColorArg, ColorResult, or ColorText pointer passed to this function*. |

Returns

An allocated ColorResult with all arguments joined together.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.
If used inside of the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros, *they will `free()` the result*. Otherwise, *you are responsible for calling `free()`*.

Examples:

colr_cat_example.c.

### 0.6.2.3.30 colr_cat

```
#define colr_cat(
              ...  ) _colr_join("", __VA_ARGS__, _ColrLastArg)
```

Join ColorArg pointers, ColorResult pointers, ColorText pointers, and strings into one long string.

To build the ColorArg pointers, it is better to use the fore(), back(), and style() macros. The ColorArgs are heap allocated, but colr_cat() will free() them for you.

To build the ColorText pointers, it is better to use the Colr() macro, along with the fore(), back(), and style() macros. The ColorTexts are heap allocated, but colr_cat() will free() them for you.

You can use ColrResult() to wrap any *allocated* string and colr_cat() will free it for you. Do not wrap static/stack-allocated strings. It will result in an "invalid free". The result of Colr_join() is an allocated ColorResult, like ColrResult() returns.

If you do not want the colr macros to free your Colr-based structs/strings for you, then you will have to call colr_to_str() on the structs and build or join the resulting strings yourself.

Parameters

| | | |
|---|---|---|
| in | ... | One or more ColorArg pointers, ColorResult pointers, ColorText pointers, or strings to join.<br>*colr_free() will be called on any ColorArg, ColorResult, or ColorText pointer passed to this function*. |

Returns

An allocated string result.
*You must `free()` the memory allocated by this function.*

See also

Colr

Examples:

back_example.c, ColorResult_example.c, colr_cat_example.c, Colr_example.c, fore_↩ example.c, simple_example.c, and style_example.c.

#### 0.6.2.3.31 Colr_center

```
#define Colr_center(
              text,
              justwidth,
              ...  )
```

**Value:**

```
ColorText_set_just( \
        Colr(text, __VA_ARGS__), \
        (ColorJustify){.method=JUST_CENTER, .width=justwidth, .padchar=' '} \
    )
```

Sets the JustifyMethod for a ColorText while allocating it.

This is like Colr_center_char(), except is uses space as the default character.

Parameters

| | | |
|---|---|---|
| in | *text* | Text to colorize. |
| in | *justwidth* | Width for justification. |
| in | ... | Fore, back, or style ColorArgs for Colr(). |

Returns

An allocated ColorText.
If used inside of the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros, *they will `free()` the result*. Otherwise, *you are responsible for calling `free()`*.

Examples:

Colr_example.c.

#### 0.6.2.3.32 Colr_center_char

```
#define Colr_center_char(
              text,
```

```
            justwidth,
            c,
            ... )
```

**Value:**

```
ColorText_set_just( \
        Colr(text, __VA_ARGS__), \
        (ColorJustify){.method=JUST_CENTER, .width=justwidth, .padchar=c} \
    )
```

Sets the JustifyMethod for a ColorText while allocating it.

Parameters

| | | |
|---|---|---|
| in | *text* | Text to colorize. |
| in | *justwidth* | Width for justification. |
| in | *c* | The character to pad with. |
| in | ... | Fore, back, or style ColorArgs for Colr(). |

Returns

An allocated ColorText.
If used inside of the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros, *they will `free()` the result*. Otherwise, *you are responsible for calling `free()`*.

See also

Colr_center

#### 0.6.2.3.33 colr_eq

```
#define colr_eq(
            a,
            b )
```

**Value:**

```
_Generic( \
        (a), \
        ArgType: ArgType_eq, \
        BasicValue: BasicValue_eq, \
        ColorArg: ColorArg_eq, \
        ColorJustify: ColorJustify_eq, \
        ColorType: ColorType_eq, \
        ColorValue: ColorValue_eq, \
        ExtendedValue: ExtendedValue_eq, \
        RGB: RGB_eq, \
        StyleValue: StyleValue_eq \
    )(a, b)
```

Calls the <type>_eq functions for the supported types.

The types for a and b must be the same.

Parameters

| | | |
|---|---|---|
| in | *a* | First supported type to compare. |
| in | *b* | Second supported type to compare. |

Returns

`true` if the values are equal, otherwise `false`.

#### 0.6.2.3.34 colr_example

```
#define colr_example(
             x )
```

**Value:**

```
_Generic( \
        (x), \
        ColorArg: ColorArg_example, \
        ColorValue: ColorValue_example \
    )(x)
```

Calls the <type>_example functions for the supported types.

This is used to create a human-friendly representation for ColorArgs or ColorValues.

Parameters

| | | |
|---|---|---|
| in | *x* | A supported type to get an example string for. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

#### 0.6.2.3.35 COLR_FMT

```
#define COLR_FMT "R"
```

Format character string suitable for use in the printf-family of functions.

This can be defined to any single-char string before including colr.h if you don't want to use the default value.

#### 0.6.2.3.36 colr_fprintf

```
#define colr_fprintf(
              ...  ) colr_printf_macro(fprintf, __VA_ARGS__)
```

Ensure colr_printf_register() has been called, and then call `fprintf`.

Will call `free()` on any ColorArg pointer, ColorResult pointer, ColorText pointer, or the strings created by them.

Attention

This feature uses a GNU extension, and is only available when COLR_GNU is defined. See the documentation for COLR_GNU.

Parameters

| | | |
|---|---|---|
| in | ... | Arguments for `fprintf`.<br>*colr_free() will be called on any ColorArg, ColorResult, or ColorText pointer passed to this function*. |

Returns

Same as `fprintf`.

#### 0.6.2.3.37 colr_free

```
#define colr_free(
              x )
```

**Value:**

```
_Generic( \
        (x), \
        ColorArg*: ColorArg_free, \
        ColorArg**: ColorArgs_array_free, \
        ColorResult*: ColorResult_free, \
        ColorText*: ColorText_free, \
        regmatch_t**: colr_free_re_matches, \
        default: _colr_free \
    )(x)
```

Calls the <type>_free functions for the supported types.

If the type is not supported, a plain `free(x)` is used.

Colr objects that have a <type>_free function will be properly released, even through a `void` pointer (as long as the `.marker` member is set, which it will be if it was created by the Colr functions/macros).

Parameters

| | | |
|---|---|---|
| in | *x* | A pointer to a supported type to free. |

Examples:

ColorResult_example.c, colr_join_example.c, and colr_replace_all_example.c.

#### 0.6.2.3.38 COLR_GNU

```
#define COLR_GNU
```

Defined when `__GNUC__` is available, to enable `statement-expressions` and `register_printf_specifier`.

There isn't a lot of information available for `register_printf_specifier` right now. There are a couple of `tutorials` out there. No `man pages` though. It `looks like` it was introduced in `glibc-2.27`.

See also

back_str_static
fore_str_static
colr_asprintf
colr_printf
colr_printf_handler
colr_printf_info
colr_printf_macro
colr_printf_register
colr_sprintf
colr_snprintf

#### 0.6.2.3.39 colr_is_empty

```
#define colr_is_empty(
            x )
```

**Value:**

```
_Generic( \
        (x), \
        ColorArg: ColorArg_is_empty, \
        ColorJustify: ColorJustify_is_empty, \
        ColorText: ColorText_is_empty, \
        ColorValue: ColorValue_is_empty \
    )(x)
```

Calls the <type>is_empty functions for the supported types.

Parameters

| | | |
|---|---|---|
| in | *x* | A supported type to build a string from. |

#### 0.6.2.3.40 colr_is_invalid

```
#define colr_is_invalid(
            x )
```

**Value:**

```
_Generic( \
        (x), \
        BasicValue: BasicValue_is_invalid, \
        ExtendedValue: ExtendedValue_is_invalid, \
        StyleValue: StyleValue_is_invalid, \
        ColorArg: ColorArg_is_invalid, \
        ColorType: ColorType_is_invalid, \
        ColorValue: ColorValue_is_invalid \
    )(x)
```

Calls the <type>is_invalid functions for the supported types.

Parameters

| | | |
|---|---|---|
| in | *x* | A supported type to build a string from. |

#### 0.6.2.3.41 colr_is_valid

```
#define colr_is_valid(
            x )
```

**Value:**

```
_Generic( \
        (x), \
        BasicValue: BasicValue_is_valid, \
        ExtendedValue: ExtendedValue_is_valid, \
        StyleValue: StyleValue_is_valid, \
        ColorArg: ColorArg_is_valid, \
        ColorType: ColorType_is_valid, \
        ColorValue: ColorValue_is_valid \
    )(x)
```

Calls the <type>is_valid functions for the supported types.

Parameters

| | | |
|---|---|---|
| in | *x* | A supported type to build a string from. |

#### 0.6.2.3.42 colr_is_valid_mblen

```
#define colr_is_valid_mblen(
            x ) ((x) && ((x) != (size_t)-1) && ((x) != (size_t)-2))
```

Checks return values from `mbrlen()` and colr_mb_len().

Parameters

| | | |
|---|---|---|
| in | *x* | A `size_t` return value to check, from `mbrlen()` or colr_mb_len(). |

Returns

`true` if at least one valid multibyte character length was detected, otherwise `false`. Invalid/incomplete multibyte sequences, or empty/ `NULL` strings will cause this macro to return `false`.

Referenced by _rainbow(), and colr_mb_len().

#### 0.6.2.3.43 colr_istr_either

```
#define colr_istr_either(
            s1,
            s2,
            s3 )
```

**Value:**

```
( \
        ((s1) && (s2) && (s3)) ? \
            (colr_istr_eq((s1), (s2)) || colr_istr_eq((s1), (s3))) : \
            false \
    )
```

Convenience macro for `!strcasecmp(s1, s2) || !strcasecmp(s1, s3)`.

Parameters

| | | |
|---|---|---|
| in | *s1* | The string to compare against the other two strings. |
| in | *s2* | The first string to compare with. |
| in | *s3* | The second string to compare with. |

Returns

1 if s1 is equal to s2 or s3, otherwise 0.

Referenced by colr_supports_rgb().

#### 0.6.2.3.44 colr_istr_eq

```
#define colr_istr_eq(
              s1,
              s2 )
```

**Value:**

```
( \
        ((s1) && (s2)) ? \
            !strcasecmp((s1), (s2)) : \
            false \
    )
```

Convenience macro for `!strcasecmp(s1, s2)`.

Parameters

| | | |
|---|---|---|
| in | *s1* | The first string to compare. |
| in | *s2* | The second string to compare. |

Returns

1 if s1 and s2 are equal, otherwise 0.

#### 0.6.2.3.45 Colr_join

```
#define Colr_join(
              joiner,
              ...  ) ColrResult(colr_join(joiner, __VA_ARGS__))
```

Joins Colr objects and strings, exactly like colr_join(), but returns an allocated ColorResult that the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros will automatically `free()` for you.

Parameters

| | | |
|---|---|---|
| in | *joiner* | What to put between the other arguments. ColorArg pointer, ColorResult pointer, ColorText pointer, or string (`char*`). |
| in | ... | Other arguments to join, with `joiner` between them. ColorArg pointers, ColorResult pointers, ColorText pointers, or strings, in any order. *colr_free() will be called on any ColorArg, ColorResult, or ColorText pointer passed to this function.* |

Returns

An allocated ColorResult.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.
If used inside of the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros, *they will `free()` the result*. Otherwise, *you are responsible for calling `free()`*.

See also

ColorResult
colr_join
colr
Colr

Examples:

ColorResult_example.c, colr_cat_example.c, colr_join_example.c, colr_replace_all_example.c, colr_replace_example.c, colr_replace_re_all_example.c, colr_replace_re_example.c, and simple_example.c.

#### 0.6.2.3.46 colr_join

```
#define colr_join(
              joiner,
              ...  ) _colr_join(joiner, __VA_ARGS__, _ColrLastArg)
```

Join ColorArg pointers, ColorText pointers, and strings by another ColorArg pointer, ColorText pointer, or string.

To build the ColorArg pointers, it is better to use the fore(), back(), and style() macros. The ColorArgs are heap allocated, but colr_join() will free() them for you.

To build the ColorText pointers, it is better to use the Colr() macro, along with the fore(), back(), and style() macros. The ColorTexts are heap allocated, but colr_join() will free() them for you.

Parameters

| | | |
|---|---|---|
| `in` | *joiner* | What to put between the other arguments. ColorArg pointer, ColorText pointer, or string. |
| `in` | ... | Other arguments to join, with `joiner` between them. ColorArg pointers, ColorText pointers, or strings, in any order. |

Returns

An allocated string.
*You must `free()` the memory allocated by this function.*

See also

colr
Colr

Examples:

ColorResult_example.c, colr_join_example.c, and simple_example.c.

#### 0.6.2.3.47 colr_length

```
#define colr_length(
              x )
```

**Value:**

```
_Generic( \
        (x), \
        ColorArg: ColorArg_length, \
        ColorResult: ColorResult_length, \
        ColorText: ColorText_length, \
        ColorValue: ColorValue_length, \
        void*: _colr_ptr_length \
    )(x)
```

Calls the <type>_length functions for the supported types.

If a `void` pointer is given, _colr_ptr_length() is called on it to determine the length.

Parameters

| | | |
|---|---|---|
| in | *x* | A supported type to build a string from. |

#### 0.6.2.3.48 Colr_ljust

```
#define Colr_ljust(
              text,
              justwidth,
              ... )
```

**Value:**

```
ColorText_set_just( \
        Colr(text, __VA_ARGS__), \
        (ColorJustify){.method=JUST_LEFT, .width=justwidth, .padchar=' '} \
    )
```

Sets the JustifyMethod for a ColorText while allocating it.

This is like Colr_ljust_char(), except is uses space as the default character.

Parameters

| | | |
|---|---|---|
| in | *text* | Text to colorize. |
| in | *justwidth* | Width for justification. |
| in | ... | Fore, back, or style ColorArgs for Colr(). |

Returns

An allocated ColorText.
If used inside of the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros, *they will `free()` the result*. Otherwise, *you are responsible for calling `free()`*.

Examples:

Colr_example.c.

#### 0.6.2.3.49 Colr_ljust_char

```
#define Colr_ljust_char(
            text,
            justwidth,
            c,
            ...  )
```

**Value:**

```
ColorText_set_just( \
        Colr(text, __VA_ARGS__), \
        (ColorJustify){.method=JUST_LEFT, .width=justwidth, .padchar=c} \
    )
```

Sets the JustifyMethod for a ColorText while allocating it.

Parameters

| | | |
|---|---|---|
| in | *text* | Text to colorize. |
| in | *justwidth* | Width for justification. |
| in | *c* | The character to pad with. |
| in | ... | Fore, back, or style ColorArgs for Colr(). |

Returns

An allocated ColorText.
If used inside of the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros, *they will `free()` the result*. Otherwise, *you are responsible for calling `free()`*.

See also

Colr_ljust

#### 0.6.2.3.50 colr_max

```
#define colr_max(
            a,
            b ) (a > b ?  a :  b)
```

Macro for `(a > b ? a : b)`.

Parameters

| | | |
|---|---|---|
| in | *a* | First value to compare. |
| in | *b* | Second value to compare. |

Returns

a if a > b, otherwise b.

Referenced by ColorText_length().

#### 0.6.2.3.51 colr_print

```
#define colr_print(
            ...  )
```

**Value:**

```
do { \
        char* _c_p_s = colr_cat(__VA_ARGS__); \
        if (!_c_p_s) break; \
        printf("%s", _c_p_s); \
        colr_free(_c_p_s); \
    } while (0)
```

Create a string from a colr_cat() call, print it to stdout (without a newline), and free it.

Parameters

| | | |
|---|---|---|
| in | ... | Arguments for colr_cat(). |

#### 0.6.2.3.52 colr_printf

```
#define colr_printf(
              ...  ) colr_printf_macro(printf, __VA_ARGS__)
```

Ensure colr_printf_register() has been called, and then call `printf`.

Will call `free()` on any ColorArg pointer, ColorResult pointer, ColorText pointer, or the strings created by them.

Attention

This feature uses a GNU extension, and is only available when COLR_GNU is defined. See the documentation for COLR_GNU.

Parameters

| | | |
|---|---|---|
| in | ... | Arguments for `printf`.<br>*colr_free() will be called on any ColorArg, ColorResult, or ColorText pointer passed to this function*. |

Returns

Same as `printf`.

Examples:

colr_printf_example.c.

#### 0.6.2.3.53 colr_printf_macro

```
#define colr_printf_macro(
              func,
              ...  )
```

**Value:**

```
__extension__({ \
        _Pragma("GCC diagnostic push"); \
        _Pragma("GCC diagnostic ignored \"-Wformat=\""); \
        _Pragma("GCC diagnostic ignored \"-Wformat-extra-args\""); \
        _Pragma("clang diagnostic push"); \
        _Pragma("clang diagnostic ignored \"-Wformat-invalid-specifier\""); \
        colr_printf_register(); \
        int _c_p_m_ret = func(__VA_ARGS__); \
        _Pragma("clang diagnostic pop"); \
        _Pragma("GCC diagnostic pop"); \
        _c_p_m_ret; \
    })
```

Calls one of the printf-family functions, with format warnings disabled for the call, and returns the result.

This function also ensures that colr_printf_register() is called, which ensures that register_printf↩ _specifier() is called one time.

Attention

This feature uses a GNU extension, and is only available when COLR_GNU is defined. See the documentation for COLR_GNU.

Parameters

| | | |
|---|---|---|
| in | *func* | The standard printf function to call, with a return type of `int`. |
| in | ... | Arguments for the printf function. |

Returns

Same as `func(...)`.

#### 0.6.2.3.54 colr_puts

```
#define colr_puts(
              ...  )
```

**Value:**

```
do { \
        char* _c_p_s = colr_cat(__VA_ARGS__); \
        if (!_c_p_s) break; \
        puts(_c_p_s); \
        colr_free(_c_p_s); \
    } while (0)
```

Create a string from a colr_cat() call, print it (with a newline), and free it.

Parameters

| | | |
|---|---|---|
| in | ... | Arguments for colr_cat(). |

Examples:

colr_cat_example.c, colr_join_example.c, and simple_example.c.

#### 0.6.2.3.55 colr_replace

```
#define colr_replace(
              s,
              target,
              repl )
```

Replace a substring in `s` with another string, ColorArg string, ColorResult string, or ColorText string.

If a string (`char*`) is used as `target` and `repl`, this is just a wrapper around colr_str_replace().

If `target` is a string (`char*`), this is a plain string-replace.

If `target` is a regex pattern (`regex_t`), it's regex match (`regmatch_t`) will be used to find a target string to replace in `s`.

If `target` is a regex match (`regmatch_t`), it's offsets will be used to find a target string in `s`.

If `target` is a `NULL`-terminated array of regex matches (`regmatch_t**`), each match will be replaced in the target string, `s`.

There is no difference between colr_replace() and colr_replace_all() when a `NULL`-terminated array of regex matches (`regmatch_t**`) is used.

If a ColorArg, ColorResult, or ColorText is used as `repl`, the appropriate colr_str_replace_<types> function is called. The function will create a string of escape-codes/text to be used as a replacement.

If `repl` is `NULL`, then an empty string (`""`) is used as the replacement, which causes the `target` string to be removed.

If you would like to replace **all** occurrences of the substring, use colr_replace_all().

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on.<br>*Must be `null`-terminated*. |
| `in` | *target* | A target string, regex pattern (`regex_t`), or regex match (`regmatch_t`) to replace in `s`. If a string is given, it *must be null-terminated*. |
| `in` | *repl* | A string, ColorArg, ColorResult, or ColorText to replace the target string with. If this is `NULL`, then an empty string is used (`""`) as the replacement.<br>*colr_free() will be called on any ColorArg, ColorResult, or ColorText pointer passed to this function*. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace_all
colr_replace_re
colr_replace_re_all
colr_str_replace
colr_str_replace_ColorArg
colr_str_replace_ColorResult
colr_str_replace_ColorText
colr_str_replace_re_pat
colr_str_replace_re_pat_ColorArg

colr_str_replace_re_pat_ColorResult
colr_str_replace_re_pat_ColorText
colr_str_replace_re_match
colr_str_replace_re_match_ColorArg
colr_str_replace_re_match_ColorResult
colr_str_replace_re_match_ColorText

Examples:

colr_replace_example.c, and simple_example.c.

#### 0.6.2.3.56 colr_replace_all

```
#define colr_replace_all(
              s,
              target,
              repl )
```

**Value:**

```
_Generic( \
        (repl), \
        char*: _Generic( \
            (target), \
                char* : colr_str_replace_all, \
                regex_t* : colr_str_replace_re_pat_all, \
                regmatch_t** : colr_str_replace_re_matches \
            ), \
        ColorArg*: _Generic( \
            (target), \
                char* : colr_str_replace_all_ColorArg, \
                regex_t* : colr_str_replace_re_pat_all_ColorArg, \
                regmatch_t** : colr_str_replace_re_matches_ColorArg \
            ), \
        ColorResult*: _Generic( \
            (target), \
                char* : colr_str_replace_all_ColorResult, \
                regex_t* : colr_str_replace_re_pat_all_ColorResult,
      \
                regmatch_t** : colr_str_replace_re_matches_ColorResult
       \
            ), \
        ColorText*: _Generic( \
            (target), \
                char* : colr_str_replace_all_ColorText, \
                regex_t* : colr_str_replace_re_pat_all_ColorText, \
                regmatch_t** : colr_str_replace_re_matches_ColorText \
            ) \
    )(s, target, repl)
```

Replace all substrings in `s` with another string, ColorArg string, ColorResult string, or ColorText string.

If a string (`char*`) is used as `target` and `repl`, this is just a wrapper around colr_str_replace().

If `target` is a string (`char*`), this is a plain string-replace.

If `target` is a regex pattern (`regex_t`), it's regex match (`regmatch_t`) will be used to find a target string to replace in `s`.

If `target` is a `NULL`-terminated array of regex matches (`regmatch_t**`), each match will be replaced in the target string, `s`.

There is no difference between colr_replace() and colr_replace_all() when a `NULL`-terminated array of regex matches (`regmatch_t**`) is used.

If a ColorArg, ColorResult, or ColorText is used as `repl`, the appropriate colr_str_replace_<types> function is called. The function will create a string of escape-codes/text to be used as a replacement.

If `repl` is `NULL`, then an empty string (`""`) is used as the replacement, which causes the `target` string to be removed.

If you would like to replace **only the first** occurrence of the substring, use colr_replace().

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on.<br>*Must be `null`-terminated*. |
| `in` | *target* | A target string, or regex pattern (`regex_t`) to replace in `s`. If a string is given, it *must be null-terminated*. |
| `in` | *repl* | A string, ColorArg, ColorResult, or ColorText to replace the target string with. If this is `NULL`, then an empty string is used (`""`) as the replacement.<br>*colr_free() will be called on any ColorArg, ColorResult, or ColorText pointer passed to this function*. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re
colr_replace_re_all
colr_str_replace_all
colr_str_replace_all_ColorArg
colr_str_replace_all_ColorResult
colr_str_replace_all_ColorText
colr_str_replace_re_pat_all
colr_str_replace_re_pat_all_ColorArg
colr_str_replace_re_pat_all_ColorResult
colr_str_replace_re_pat_all_ColorText

Examples:

colr_replace_all_example.c, and simple_example.c.

#### 0.6.2.3.57 colr_replace_re

```
#define colr_replace_re(
              s,
              target,
              repl,
              flags )
```

**Value:**

```
_Generic( \
        (repl), \
        char*: colr_str_replace_re, \
        ColorArg*: colr_str_replace_re_ColorArg, \
        ColorResult*: colr_str_replace_re_ColorResult, \
        ColorText*: colr_str_replace_re_ColorText \
    )(s, target, repl, flags)
```

Replace a regex pattern string (`char*`) in `s` with another string, ColorArg string, ColorResult string, or ColorText string.

If a string (`char*`) is used as `repl`, this is just a wrapper around colr_str_replace_re().

If a ColorArg, ColorResult, or ColorText is used as `repl`, the appropriate colr_str_replace_re_<type> function is called. The function will create a string of escape-codes/text to be used as a replacement.

If `repl` is `NULL`, then an empty string (`""`) is used as the replacement, which causes the `target` string to be removed.

If you would like to replace **all** occurrences of the substring, use colr_replace_re_all().

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on.<br>*Must be `null`-terminated.* |
| `in` | *target* | A regex pattern string (`char*`), regex pattern (`regex_t`), or regex match (`regmatch_t`) to replace in `s`.<br>If a string is given, it *must be null-terminated*. |
| `in` | *repl* | A string, ColorArg, ColorResult, or ColorText to replace the target string with. If this is `NULL`, then an empty string is used (`""`) as the replacement.<br>*colr_free() will be called on any ColorArg, ColorResult, or ColorText pointer passed to this function.* |
| `in` | *flags* | Flags for `regcomp()`. `REG_EXTENDED` is always used, whether flags are provided or not. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

colr_replace
colr_replace_all
colr_replace_re_all
colr_str_replace_re
colr_str_replace_re_ColorArg
colr_str_replace_re_ColorResult
colr_str_replace_re_ColorText

Examples:

colr_replace_re_example.c, and simple_example.c.

#### 0.6.2.3.58 colr_replace_re_all

```
#define colr_replace_re_all(
            s,
            target,
            repl,
            flags )
```

**Value:**

```
_Generic( \
        (repl), \
        char*: colr_str_replace_re_all, \
        ColorArg*: colr_str_replace_re_all_ColorArg, \
        ColorResult*: colr_str_replace_re_all_ColorResult, \
        ColorText*: colr_str_replace_re_all_ColorText \
    )(s, target, repl, flags)
```

Replace all matches to a regex pattern string (`char*`) in `s` with another string, ColorArg string, ColorResult string, or ColorText string.

If a string (`char*`) is used as `repl`, this is just a wrapper around colr_str_replace_re_all().

If a ColorArg, ColorResult, or ColorText is used as `repl`, the appropriate colr_str_replace_re_<type> function is called. The function will create a string of escape-codes/text to be used as a replacement.

If `repl` is `NULL`, then an empty string (`""`) is used as the replacement, which causes the `target` string to be removed.

If you would like to replace **only the first** occurrence of the substring, use colr_replace_re().

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on.<br>*Must be `null`-terminated*. |
| in | *target* | A regex pattern string (`char*`), regex pattern (`regex_t`), or regex match (`regmatch_t`) to replace in `s`.<br>If a string is given, it *must be null-terminated*. |
| in | *repl* | A string, ColorArg, ColorResult, or ColorText to replace the target string with. If this is `NULL`, then an empty string is used (`""`) as the replacement.<br>*colr_free() will be called on any ColorArg, ColorResult, or ColorText pointer passed to this function*. |
| in | *flags* | Flags for `regcomp()`. `REG_EXTENDED` is always used, whether flags are provided |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_all
colr_replace_re
colr_str_replace_re
colr_str_replace_re_ColorArg
colr_str_replace_re_ColorResult
colr_str_replace_re_ColorText

Examples:

colr_replace_re_all_example.c.

#### 0.6.2.3.59 colr_repr

```
#define colr_repr(
              x )
```

**Value:**

```
_Generic( \
        (x), \
        ColorArg: ColorArg_repr, \
        ColorArg**: ColorArgs_array_repr, \
        ColorJustify: ColorJustify_repr, \
        ColorJustifyMethod: ColorJustifyMethod_repr, \
        ColorResult: ColorResult_repr, \
        ColorText: ColorText_repr, \
        ColorValue: ColorValue_repr, \
        ArgType: ArgType_repr, \
        ColorType: ColorType_repr, \
        BasicValue: BasicValue_repr, \
        ExtendedValue: ExtendedValue_repr, \
        RGB: RGB_repr, \
        StyleValue: StyleValue_repr, \
        TermSize: TermSize_repr, \
        const char*: colr_str_repr, \
        char*: colr_str_repr, \
        const char: colr_char_repr, \
        char: colr_char_repr, \
        void*: _colr_ptr_repr \
    )(x)
```

Transforms several ColrC objects into their string representations.

Uses _Generic (C11 standard) to dynamically dispatch to the proper `*_repr` functions.

If a regular string is passed in, it will be escaped and you must still `free()` the result.

Supported Types:

- ColorArg
- ColorJustify
- ColorJustifyMethod
- ColorText
- ColorValue
- ArgType
- ColorType
- BasicValue
- ExtendedValue
- RGB
- StyleValue
- TermSize
- char*
- char

Parameters

| | | |
|---|---|---|
| in | *x* | A value with one of the supported types to transform into a string. |

Returns

Stringified representation of what was passed in.
*You must `free()` the memory allocated by this function.*

Referenced by ColorArgs_array_repr(), colr_printf_handler(), and colr_str_mb_len().

#### 0.6.2.3.60 Colr_rjust

```
#define Colr_rjust(
              text,
              justwidth,
              ...  )
```

**Value:**

```
ColorText_set_just( \
        Colr(text, __VA_ARGS__), \
        (ColorJustify){.method=JUST_RIGHT, .width=justwidth, .padchar=' '} \
    )
```

Sets the JustifyMethod for a ColorText while allocating it.

This is like Colr_rjust_char(), except is uses space as the default character.

Parameters

| | | |
|---|---|---|
| in | *text* | Text to colorize. |
| in | *justwidth* | Width for justification. |
| in | ... | Fore, back, or style ColorArgs for Colr(). |

Returns

An allocated ColorText.
If used inside of the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros, *they will `free()` the result*. Otherwise, *you are responsible for calling `free()`*.

Examples:

Colr_example.c.

#### 0.6.2.3.61 Colr_rjust_char

```
#define Colr_rjust_char(
            text,
            justwidth,
            c,
            ...  )
```

**Value:**

```
ColorText_set_just( \
        Colr(text, __VA_ARGS__), \
        (ColorJustify){.method=JUST_RIGHT, .width=justwidth, .padchar=c} \
    )
```

Sets the JustifyMethod for a ColorText while allocating it.

Parameters

| | | |
|---|---|---|
| in | *text* | Text to colorize. |
| in | *justwidth* | Width for justification. |
| in | *c* | The character to pad with. |
| in | ... | Fore, back, or style ColorArgs for Colr(). |

Returns

An allocated ColorText.
If used inside of the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros, *they will `free()` the result*. Otherwise, *you are responsible for calling `free()`*.

See also

Colr_rjust

#### 0.6.2.3.62 colr_snprintf

```
#define colr_snprintf(
              ...  ) colr_printf_macro(snprintf, __VA_ARGS__)
```

Ensure colr_printf_register() has been called, and then call `snprintf`.

Will call `free()` on any ColorArg pointer, ColorResult pointer, ColorText pointer, or the strings created by them.

Attention

This feature uses a GNU extension, and is only available when COLR_GNU is defined. See the documentation for COLR_GNU.

Parameters

| | | |
|---|---|---|
| `in` | ... | Arguments for `snprintf`.<br>*colr_free() will be called on any ColorArg, ColorResult, or ColorText pointer passed to this function.* |

Returns

Same as `snprintf`.

Examples:

colr_printf_example.c.

#### 0.6.2.3.63 colr_sprintf

```
#define colr_sprintf(
              ...  ) colr_printf_macro(sprintf, __VA_ARGS__)
```

Ensure colr_printf_register() has been called, and then call `sprintf`.

Will call `free()` on any ColorArg pointer, ColorResult pointer, ColorText pointer, or the strings created by them.

Attention

This feature uses a GNU extension, and is only available when COLR_GNU is defined. See the documentation for COLR_GNU.

Parameters

| | | |
|---|---|---|
| in | ... | Arguments for `sprintf`.<br>*colr_free() will be called on any ColorArg, ColorResult, or ColorText pointer passed to this function.* |

Returns

Same as `sprintf`.

Examples:

colr_printf_example.c.

#### 0.6.2.3.64 colr_str_either

```
#define colr_str_either(
            s1,
            s2,
            s3 ) (colr_str_eq(s1, s2) || colr_str_eq(s1, s3))
```

Convenience macro for `!strcmp(s1, s2) || !strcmp(s1, s3)`.

Parameters

| | | |
|---|---|---|
| in | *s1* | The string to compare against the other two strings. |
| in | *s2* | The first string to compare with. |
| in | *s3* | The second string to compare with. |

Returns

`1` if `s1` is equal to `s2` or `s3`, otherwise `0`.

#### 0.6.2.3.65 colr_str_eq

```
#define colr_str_eq(
            s1,
            s2 )
```

**Value:**

```
( \
        ((s1) && (s2)) ? !strcmp((s1), (s2)) : false \
    )
```

Convenience macro for `!strcmp(s1, s2)`.

Parameters

| | | |
|---|---|---|
| in | *s1* | The first string to compare. |
| in | *s2* | The second string to compare. |

Returns

`1` if `s1` and `s2` are equal, otherwise `0`.

Referenced by ColorResult_eq(), and RGB_from_str().

#### 0.6.2.3.66 colr_to_str

```
#define colr_to_str(
              x )
```

**Value:**

```
_Generic( \
        (x), \
        ArgType: ArgType_to_str, \
        BasicValue: BasicValue_to_str, \
        ColorArg: ColorArg_to_esc, \
        ColorResult: ColorResult_to_str, \
        ColorText: ColorText_to_str, \
        ColorType: ColorType_to_str, \
        ExtendedValue: ExtendedValue_to_str, \
        StyleValue: StyleValue_to_str, \
        RGB: RGB_to_str, \
        void*: _colr_ptr_to_str \
    )(x)
```

Calls the <type>_to_str functions for the supported types.

If a string is given, it is duplicated like `strdup()`.

Parameters

| | | |
|---|---|---|
| in | *x* | A supported type to build a string from. |

Returns

An allocated string from the type's `*_to_str()` function.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

Examples:

ColorResult_example.c.

#### 0.6.2.3.67 Colra

```
#define Colra(
              text,
              ...  ) ColorText_from_values(text, __VA_ARGS__, _ColrLastArg)
```

Returns an initialized stack-allocated ColorText.

If this ColorText is manually stored on the heap, and then sent through the colr macros, it's Color↩ Args will be free'd. You cannot use the same ColorText twice inside the colr macros/functions.

Attention

The result cannot be used inside the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros, because you must not call `free()` on it.

Parameters

| | | |
|---|---|---|
| `in` | *text* | String to colorize/style. |
| `in` | ... | No more than 3 ColorArg pointers for fore, back, and style in any order. |

Returns

An initialized ColorText.

See also

Colr

#### 0.6.2.3.68 ColrResult

```
#define ColrResult(
              s ) ColorResult_to_ptr(ColorResult_new(s))
```

Wraps an allocated string in a ColorResult, which marks it as "freeable" in the colr macros.

Parameters

| | | |
|---|---|---|
| `in` | *s* | An allocated string. |

Returns

An allocated ColorResult.

If used inside of the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros, *they will `free()` the result*. Otherwise, *you are responsible for calling `free()`*.

Examples:

ColorResult_example.c.

#### 0.6.2.3.69 ext

```
#define ext(
            x ) ((ExtendedValue)x)
```

Casts to ExtendedValue (`unsigned char`).

Parameters

| | | |
|---|---|---|
| in | *x* | Value to cast to `unsigned char`/`ExtendedValue`. |

Returns

An ExtendedValue.

See also

fore
back
colr
Colr
ext_hex
ext_hex_or
ext_rgb
ext_RGB

Examples:

back_example.c, colr_cat_example.c, fore_example.c, and simple_example.c.

Referenced by ExtendedValue_from_BasicValue(), and ExtendedValue_from_RGB().

#### 0.6.2.3.70 ext_hex

```
#define ext_hex(
            s ) ext_hex_or(s, ext(0))
```

Like hex(), but force a conversion to the closest ExtendedValue (256-colors).

Parameters

| | | |
|---|---|---|
| in | *s* | A hex string to convert. |

Returns

The closest matching ExtendedValue, or 0 for bad hex strings.

See also

ext
ext_hex_or
hex
hex_or

Examples:

back_example.c, Colr_example.c, colr_join_example.c, and simple_example.c.

#### 0.6.2.3.71 ext_hex_or

```
#define ext_hex_or(
            s,
            default_value ) ExtendedValue_from_hex_default(s, default_value)
```

Like hex_or(), but force a conversion to the closest ExtendedValue (256-colors).

This is a convenience macro for ExtendedValue_from_hex_default().

Parameters

| | | |
|---|---|---|
| in | *s* | A hex string to convert. |
| in | *default_value* | ExtendedValue to use if the hex string is not valid. |

Returns

The closest matching ExtendedValue, or `default_value` for bad hex strings.

See also

ext
ext_hex
hex
hex_or

Examples:

back_example.c.

#### 0.6.2.3.72 EXT_INVALID

```
#define EXT_INVALID COLOR_INVALID
```

Alias for COLOR_INVALID.

All color values share an _INVALID member with the same value, so:

```
COLOR_INVALID == BASIC_INVALID == EXT_INVALID == STYLE_INVALID
```

Referenced by ExtendedValue_from_BasicValue(), ExtendedValue_from_esc(), and ExtendedValue_from_str().

#### 0.6.2.3.73 EXT_INVALID_RANGE

```
#define EXT_INVALID_RANGE COLOR_INVALID_RANGE
```

Possible error return value for ExtendedValue_from_str() or ExtendedValue_from_esc().

This is just an alias for COLOR_INVALID_RANGE.

```
COLOR_INVALID_RANGE == BASIC_INVALID_RANGE ==
      EXT_INVALID_RANGE == STYLE_INVALID_RANGE
```

Referenced by ExtendedValue_from_esc(), and ExtendedValue_from_str().

#### 0.6.2.3.74 ext_rgb

```
#define ext_rgb(
              r,
              g,
              b ) ExtendedValue_from_RGB((RGB){.red=r, .green=g, .blue=b})
```

Creates the closest matching ExtendedValue from separate red, green, and blue values.

This is short-hand for `ExtendedValue_from_RGB((RGB){r, g, b})`.

Parameters

| | | |
|---|---|---|
| in | *r* | The red value. |
| in | *g* | The green value. |
| in | *b* | The blue value. |

Returns

An ExtendedValue that closely matches the RGB value.

See also

ExtendedValue_from_RGB
RGB_to_term_RGB

Examples:

ColorResult_example.c, and Colr_example.c.

#### 0.6.2.3.75 ext_RGB

```
#define ext_RGB(
            rgbval ) ExtendedValue_from_RGB(rgbval)
```

Creates the closest matching ExtendedValue from an RGB value.

This is short-hand for `ExtendedValue_from_RGB(rgbval)`.

Parameters

| | | |
|---|---|---|
| in | *rgbval* | The RGB value to use. |

Returns

An ExtendedValue that closely matches the RGB value.

See also

ExtendedValue_from_RGB
RGB_to_term_RGB

#### 0.6.2.3.76 fore

```
#define fore(
            x ) ColorArg_to_ptr(fore_arg(x))
```

Create a fore color suitable for use with the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros.

Technically, this macro accepts BasicValues, ExtendedValues, or RGB structs. However, for some of these you should be using the macros that create those things.

BasicValues can be used by their names (RED, YELLOW, etc.).

ExtendedValues can be created on the fly with ext().

RGB structs can be easily created with rgb().

Color names (`char*`) can be passed to generate the appropriate color value.

Parameters

| | | |
|---|---|---|
| in | *x* | A BasicValue, ExtendedValue, or RGB struct to use for the color value. |

Returns

A pointer to a heap-allocated ColorArg struct.
If used inside of the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros, *they will `free()` the result*. Otherwise, *you are responsible for calling `free()`*.

See also

fore_arg
fore_str
colr
Colr

Examples:

back_example.c, ColorResult_example.c, colr_cat_example.c, Colr_example.c, colr_join_example.c, colr_printf_example.c, colr_replace_all_example.c, colr_replace_example.c, colr_replace_re_all_example.c, colr_replace_re_example.c, fore_example.c, and simple_example.c.

#### 0.6.2.3.77 fore_arg

```
#define fore_arg(
              x )
```

**Value:**

```
_Generic( \
        (x), \
        char* : ColorArg_from_str, \
        RGB: ColorArg_from_RGB, \
        BasicValue: ColorArg_from_BasicValue, \
        ExtendedValue: ColorArg_from_ExtendedValue, \
        StyleValue: ColorArg_from_StyleValue \
    )(FORE, x)
```

Uses ColorArg_from_<type> to build a ColorArg with the appropriate color type, based on the type of it's argument.

Uses `_Generic` (C11 standard) to dynamically create a ColorArg. This is used by the fore() macro.

Parameters

| | | |
|---|---|---|
| in | *x* | `BasicValue`, `Extended` (`unsigned char`), RGB struct, or string (color name) for fore color. |

Returns

A ColorArg with the FORE type set, and it's `.value.type` set for the appropriate color type/value. For invalid values the `.value.type` may be set to TYPE_INVALID.

See also

fore
fore_str

#### 0.6.2.3.78 fore_str

```
#define fore_str(
            x ) ColorArg_to_esc(fore_arg(x))
```

Return just the escape code string for a fore color.

Parameters

| | | |
|---|---|---|
| in | *x* | A BasicValue, ExtendedValue, or RGB struct. |

Returns

An allocated string.
*You must `free()` the memory allocated by this function.*

See also

fore
fore_arg

#### 0.6.2.3.79 fore_str_static

```
#define fore_str_static(
            x )
```

**Value:**

```
__extension__ ({ \
        __typeof(x) _fss_val = x; \
        ColorArg _fss_carg = fore_arg(_fss_val); \
        size_t _fss_len = ColorArg_length(_fss_carg); \
        char* _fss_codes = alloca(_fss_len); \
        ColorArg_to_esc_s(_fss_codes, _fss_carg); \
        _fss_codes; \
    })
```

Creates a stack-allocated escape code string (`char*`) for a fore color.

These are not constant strings, but they are stored on the stack. A `Statement Expression` is used to build a string of the correct length and content using ColorArg_to_esc_s().

Attention

This feature uses a GNU extension, and is only available when COLR_GNU is defined. See the documentation for COLR_GNU.

Warning

This uses `alloca` to reserve space on the stack inside of a `Statement Expression`. A `Variable Length Array` will not work inside of a statement expression. If the call causes a stack overflow, program behavior is undefined. See previous links, and `here`.

You can also create stack-allocated escape code strings using format_fg(), format_fgx(), format_fg_RGB(), and format_fg_RGB_term().

Parameters

| | | |
|---|---|---|
| in | *x* | A BasicValue, ExtendedValue, or RGB value. |

Returns

A stack-allocated escape code string.

See also

back_str_static
style_str_static
format_fg
format_bg

### 0.6.2.3.80 hex

```
#define hex(
            s ) hex_or(s, rgb(0, 0, 0))
```

Use RGB_from_hex_default() to create an RGB value.

Parameters

| | | |
|---|---|---|
| in | *s* | A hex string to convert. |

Returns

A valid RGB value, or `rgb(0, 0, 0)` for bad hex strings.

See also

hex_or
ext_hex
ext_hex_or

Examples:

back_example.c, colr_join_example.c, and simple_example.c.

#### 0.6.2.3.81 hex_or

```
#define hex_or(
              s,
              default_rgb ) RGB_from_hex_default(s, default_rgb)
```

Use RGB_from_hex_default() to create an RGB value.

Parameters

| | | |
|---|---|---|
| in | *s* | A hex string to convert. |
| in | *default_rgb* | Default RGB value to use if the hex string is not valid. |

Returns

A valid RGB value, or `default_rgb` for bad hex strings.

See also

hex
ext_hex
ext_hex_or

Examples:

back_example.c.

#### 0.6.2.3.82 if_not_asprintf

```
#define if_not_asprintf(
              ...  ) if (asprintf(__VA_ARGS__) < 1)
```

Convenience macro for checking asprintf's return value.

Should be followed by a block of code.

Note: asprintf returns `-1` for errors, but `0` is a valid return (`0` bytes written to the string). The string will be untouched (may be `NULL` if it was initialized as `NULL`)

Parameters

| | | |
|---|---|---|
| in | ... | Arguments for asprintf. |

#### 0.6.2.3.83 rgb

```
#define rgb(
              r,
              g,
              b ) ((RGB){.red=r, .green=g, .blue=b})
```

Creates an anonymous RGB struct for use in function calls.

Parameters

| | | |
|---|---|---|
| in | *r* | unsigned char Red value. |
| in | *g* | unsigned char Blue value. |
| in | *b* | unsigned char Green value. |

Returns

An RGB struct.

See also

rgb_safe

Examples:

back_example.c, colr_cat_example.c, colr_join_example.c, fore_example.c, and simple_example.c.

Referenced by ExtendedValue_from_hex(), rainbow_step(), RGB_from_hex_default(), RGB_grayscale(), RGB_inverted(), and RGB_monochrome().

#### 0.6.2.3.84 style

```
#define style(
              x ) ColorArg_to_ptr(style_arg(x))
```

Create a style suitable for use with the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros.

This macro accepts strings (style names) and StyleValues.

Style names (char*) can be passed to generate the appropriate style value.

Parameters

| | | |
|---|---|---|
| in | *x* | A StyleValue. |

Returns

A pointer to a heap-allocated ColorArg struct.
If used inside of the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros, *they will `free()` the result*. Otherwise, *you are responsible for calling `free()`*.

See also

style_arg
style_str
colr
Colr

Examples:

ColorResult_example.c, colr_cat_example.c, Colr_example.c, colr_join_example.c, colr_printf_example.c, colr_replace_all_example.c, colr_replace_example.c, colr_replace_re_all_example.c, colr_replace_re_example.c, simple_example.c, and style_example.c.

#### 0.6.2.3.85 style_arg

```
#define style_arg(
              x )
```

**Value:**

```
_Generic( \
        (x), \
        char* : ColorArg_from_str, \
        StyleValue: ColorArg_from_StyleValue \
    )(STYLE, x)
```

Uses ColorArg_from_StyleValue to build a ColorArg with the appropriate color type/value.

Parameters

| | | |
|---|---|---|
| in | *x* | `StyleValue` for the style. |

Returns

A ColorArg with the STYLE type set, and it's `.value.type` set for the appropriate color type/value. For invalid values the `.value.type` may be set to TYPE_INVALID.

See also

style
style_str

#### 0.6.2.3.86 STYLE_LEN_MIN

```
#define STYLE_LEN_MIN 5
```

Minimum length for the shortest style escape code, including "\0".

Use STYLE_LEN for allocation.

#### 0.6.2.3.87 style_str

```
#define style_str(
              x ) ColorArg_to_esc(style_arg(x))
```

Return just the escape code string for a style.

Parameters

| | | |
|---|---|---|
| in | *x* | StyleValue to use. |

Returns

An allocated string.
*You must `free()` the memory allocated by this function.*

See also

style
style_arg

#### 0.6.2.3.88 style_str_static

```
#define style_str_static(
              x )
```

**Value:**

```
(x == RESET_ALL ? "\x1b[0m" : \
    (x ==BRIGHT ? "\x1b[1m" : \
    (x == DIM ? "\x1b[2m" : \
    (x == ITALIC ? "\x1b[3m" : \
    (x == UNDERLINE ? "\x1b[4m" : \
    (x == FLASH ? "\x1b[5m" : \
```

```
    (x == HIGHLIGHT ? "\x1b[7m" : \
    (x == STRIKETHRU ? "\x1b[9m" : \
    (x == NORMAL ? "\x1b[22m" : \
    (x == FRAME ? "\x1b[51m" : \
    (x == ENCIRCLE ? "\x1b[52m" : \
    (x == OVERLINE ? "\x1b[53m" : "\x1b[" colr_macro_str(x) "m" \
    ))))))))))))
```

A less-flexible style_str() that returns a static escape code string for a style.

This macro function does not accept style names. Only `StyleValue` and literal `int` values are accepted.

The resulting expression will be optimized into a constant static string.

Parameters

| | | |
|---|---|---|
| in | *x* | A StyleValue to use. |

Returns

A stack-allocated string.

See also

fore_str_static
back_str_static
format_fg
format_bg

#### 0.6.2.3.89 while_colr_va_arg

```
#define while_colr_va_arg(
              ap,
              vartype,
              x ) while (x = va_arg(ap, vartype), !_colr_is_last_arg(x))
```

Construct a while-loop over a `va_list`, where the last argument is expected to be _ColrLastArg, or a pointer to a _ColrLastArg_s with the same values as _ColrLastArg.

Parameters

| | | |
|---|---|---|
| in | *ap* | The `va_list` to use. |
| in | *vartype* | Expected type of the argument. |
| in | *x* | The variable to assign to (usually `arg`). |

Referenced by _colr_join(), _colr_join_size(), ColorText_from_values(), and ColorText_set_values().

#### 0.6.2.4 Typedef Documentation

##### 0.6.2.4.1 RGB_fmter

```
typedef void(* RGB_fmter) (char *out, RGB rgb)
```

A function type that knows how to fill a string with an rgb escape code.

#### 0.6.2.5 Enumeration Type Documentation

##### 0.6.2.5.1 BasicValue

```
enum BasicValue
```

Basic color values, with a few convenience values for extended colors.

#### 0.6.2.6 Function Documentation

##### 0.6.2.6.1 _colr_free()

```
void _colr_free (
            void * p )
```

Calls Colr `*_free()` functions for Colr objects, otherwise just calls `free()`.

You should use the colr_free() macro instead.

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| `in` | *p* | Pointer to a heap-allocated object. |

##### 0.6.2.6.2 _colr_is_last_arg()

```
bool _colr_is_last_arg (
            void * p )
```

Determines if a void pointer is _ColrLastArg (the last-arg-marker).

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| in | *p* | The pointer to check. |

Returns

`true` if the pointer is _ColrLastArg, otherwise `false`.

#### 0.6.2.6.3 _colr_join()

```
char* _colr_join (
            void * joinerp,
             ...  )
```

Joins ColorArgs, ColorTexts, and strings (`char*`) into one long string separated by it's first argument.

This will free() any `ColorArg`s, `ColorResult`s, or `ColorText`s that are passed in. It is backing the colr_cat(), colr_join(), Colr_cat(), and Colr_join() macros, and enables easy throw-away color values.

Any plain strings that are passed in are left alone. It is up to the caller to free those. ColrC only manages the temporary Colr-based objects needed to build up these strings.

You should use colr_cat(), colr_join(), Colr_cat(), and Colr_join() macros instead.

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| in | *joinerp* | The joiner (any ColorArg, ColorResult, ColorText, or string). |
| in | ... | Zero or more ColorArgs, ColorResults, ColorTexts, or strings to join by the joiner. |

Returns

An allocated string with mixed escape codes/strings. CODE_RESET_ALL is appended to all ColorText arguments. This allows easy part-colored messages.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*. Also, `NULL` will be returned if `joinerp` is `NULL`.

#### 0.6.2.6.4 _colr_join_array_length()

```
size_t _colr_join_array_length (
            void * ps )
```

Determine the length of a NULL-terminated array of strings (char*), ColorArgs, ColorResults, or ColorTexts.

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| in | *ps* | A NULL-terminated array of ColorArgs, ColorResults, ColorTexts, or strings (char*). |

Returns

The number of items (before NULL) in the array.

Referenced by colr_join_array().

#### 0.6.2.6.5 _colr_join_arrayn_size()

```
size_t _colr_join_arrayn_size (
            void * joinerp,
            void * ps,
            size_t count )
```

Get the size in bytes needed to join an array of strings (char*), ColorArgs, ColorResults, or ColorTexts by another string (char*), ColorArg, ColorResult, or ColorText.

This is used to allocate memory in the _colr_join_array() function.

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| in | *joinerp* | The joiner (any ColorArg, ColorResult, ColorText, or string). |
| in | *ps* | An array of pointers to ColorArgs, ColorResults, ColorTexts, or strings. The array must have NULL as the last item if count is greater than the total number of items. |
| in | *count* | Total number of items in the array. |

Returns

The number of bytes needed to allocate the result of colr_join_arrayn(), possibly 0.

See also

colr
colr_join
colr_join_array

Referenced by colr_join_arrayn().

#### 0.6.2.6.6 _colr_join_size()

```
size_t _colr_join_size (
            void * joinerp,
            va_list args )
```

Parse arguments, just as in _colr_join(), but only return the size needed to allocate the resulting string.

This allows _colr_join() to allocate once, instead of reallocating for each argument that is passed.

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| in | *joinerp* | The joiner (any ColorArg, ColorText, or string (char*)). |
| in | *args* | A va_list with zero or more ColorArgs, ColorTexts, or strings (char*) to join. |

Returns

The length (in bytes) needed to allocate a string built with _colr_cat(). This function will return 0 if joinerp is NULL/empty). Except for 0, it will never return anything less than CODE_RE↩ SET_LEN.

See also

_colr

Referenced by _colr_join().

#### 0.6.2.6.7 _colr_ptr_length()

```
size_t _colr_ptr_length (
            void * p )
```

Get the size, in bytes, needed to convert a ColorArg, ColorResult, ColorText, or string (`char*`) into a string.

This is used in the variadic _colr* functions.

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| `in` | *p* | A ColorArg pointer, ColorResult pointer, ColorText pointer, or string (`char*`). |

Returns

The length needed to convert the object into a string (`strlen() + 1` for strings).

Referenced by _colr_join_arrayn_size(), and _colr_join_size().

#### 0.6.2.6.8 _colr_ptr_repr()

```
char* _colr_ptr_repr (
            void * p )
```

Determine what kind of pointer is being passed, and call the appropriate <type>_repr function to obtain an allocated string representation.

You should use colr_repr() instead.

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| `in` | *p* | A ColorArg pointer, ColorResult pointer, ColorText pointer, or string. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

colr_repr

#### 0.6.2.6.9 _colr_ptr_to_str()

```
char* _colr_ptr_to_str (
            void * p )
```

Determine what kind of pointer is being passed, and call the appropriate <type>_to_str function to obtain an allocated string.

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| in | *p* | A ColorArg pointer, ColorResult pointer, ColorText pointer, or string. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

#### 0.6.2.6.10 _rainbow()

```
char* _rainbow (
            RGB_fmter fmter,
            const char * s,
            double freq,
            size_t offset,
            size_t spread )
```

Handles multibyte character string (`char*`) conversion and character iteration for all of the rainbow_ functions.

Warning

This is for internal use only.

Parameters

| | | |
|---|---|---|
| in | *fmter* | A formatter function (RGB_fmter) that can create escape codes from RGB values. |
| in | *s* | The string to "rainbowize".<br>Input *must be `null`-terminated*. |
| in | *freq* | The "tightness" for colors. |
| in | *offset* | The starting offset into the rainbow. |
| in | *spread* | Number of characters per color. |

Returns

An allocated string (`char*`) with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

Referenced by rainbow_bg(), rainbow_bg_term(), rainbow_fg(), and rainbow_fg_term().

#### 0.6.2.6.11 ArgType_eq()

```
bool ArgType_eq (
            ArgType a,
            ArgType b )
```

Compares two ArgTypes.

This is used to implement colr_eq().

Parameters

| | | |
|---|---|---|
| in | *a* | The first ArgType to compare. |
| in | *b* | The second ArgType to compare. |

Returns

`true` if they are equal, otherwise `false`.

#### 0.6.2.6.12 ArgType_repr()

```
char* ArgType_repr (
            ArgType type )
```

Creates a string (`char*`) representation of a ArgType.

Parameters

| | | |
|---|---|---|
| in | *type* | An ArgType to get the type from. |

Returns

A pointer to an allocated string.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

ArgType

Referenced by ColorArg_repr().

#### 0.6.2.6.13 ArgType_to_str()

```
char* ArgType_to_str (
            ArgType type )
```

Creates a human-friendly string (`char*`) from an ArgType.

Parameters

| | | |
|---|---|---|
| `in` | *type* | An ArgType to get the type from. |

Returns

A pointer to an allocated string.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

ArgType

Referenced by ColorArg_example().

#### 0.6.2.6.14 BasicValue_eq()

```
bool BasicValue_eq (
            BasicValue a,
            BasicValue b )
```

Compares two BasicValues.

This is used to implement colr_eq().

Parameters

| | | |
|---|---|---|
| `in` | *a* | The first BasicValue to compare. |
| `in` | *b* | The second BasicValue to compare. |

Returns

`true` if they are equal, otherwise `false`.

See also

BasicValue

#### 0.6.2.6.15 BasicValue_from_esc()

```
BasicValue BasicValue_from_esc (
            const char * s )
```

Convert an escape-code string (`char*`) to an actual BasicValue enum value.

Parameters

| | | |
|---|---|---|
| `in` | *s* | Escape-code string.<br>*Must be `null`-terminated.* |

Return values

| | |
|---|---|
| *BasicValue* | value on success. |
| *BASIC_INVALID* | on error (or if `s` is `NULL`). |
| *BASIC_INVALID_RANGE* | if the code number was outside of the range `0-255`. |

See also

BasicValue

#### 0.6.2.6.16 BasicValue_from_str()

```
BasicValue BasicValue_from_str (
            const char * arg )
```

Convert named argument to an actual BasicValue enum value.

Parameters

| | | |
|---|---|---|
| `in` | *arg* | Color name to find the BasicValue for. |

Returns

BasicValue value on success, or BASIC_INVALID on error.

See also

BasicValue

#### 0.6.2.6.17 BasicValue_is_invalid()

```
bool BasicValue_is_invalid (
            BasicValue bval )
```

Determines whether a BasicValue is invalid.

Parameters

| | | |
|---|---|---|
| in | *bval* | A BasicValue to check. |

Returns

`true` if the value is considered invalid, otherwise `false`.

See also

BasicValue

Referenced by ExtendedValue_from_BasicValue().

#### 0.6.2.6.18 BasicValue_is_valid()

```
bool BasicValue_is_valid (
            BasicValue bval )
```

Determines whether a BasicValue is valid.

Parameters

| | | |
|---|---|---|
| in | *bval* | A BasicValue to check. |

Returns

`true` if the value is considered valid, otherwise `false`.

See also

BasicValue

#### 0.6.2.6.19 BasicValue_repr()

```
char* BasicValue_repr (
            BasicValue bval )
```

Creates a string (`char*`) representation of a BasicValue.

Parameters

| | | |
|---|---|---|
| `in` | *bval* | A BasicValue to get the value from. |

Returns

A pointer to an allocated string.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

BasicValue

#### 0.6.2.6.20 BasicValue_to_ansi()

```
int BasicValue_to_ansi (
            ArgType type,
            BasicValue bval )
```

Converts a fore/back BasicValue to the actual ansi code number.

Parameters

| | | |
|---|---|---|
| `in` | *type* | ArgType (FORE/BACK). |
| `in` | *bval* | BasicValue to convert. |

Returns

An integer usable with basic escape code fore/back colors.

See also

BasicValue

Referenced by format_bg(), and format_fg().

#### 0.6.2.6.21 BasicValue_to_str()

```
char* BasicValue_to_str (
            BasicValue bval )
```

Create a human-friendly string (`char*`) representation for a BasicValue.

Parameters

| | | |
|---|---|---|
| in | *bval* | BasicValue to get the name for. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

BasicValue

#### 0.6.2.6.22 ColorArg_empty()

```
ColorArg ColorArg_empty (
            void  )
```

Create a ColorArg with ARGTYPE_NONE and ColorValue.type.TYPE_NONE.

This is used to pass "empty" fore/back/style args to the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros, where `NULL` may have a different meaning for users of the ColorArg.

Returns

`(ColorArg){.type=ARGTYPE_NONE, .value.type=TYPE_NONE}`

See also

ColorArg_is_empty
ColorValue_empty

#### 0.6.2.6.23 ColorArg_eq()

```
bool ColorArg_eq (
            ColorArg a,
            ColorArg b )
```

Compares two ColorArg structs.

They are considered "equal" if their `.type` and `.value` match.

Parameters

| | | |
|---|---|---|
| in | *a* | First ColorArg to compare. |
| in | *b* | Second ColorArg to compare. |

Returns

`true` if they are equal, otherwise `false`.

See also

ColorArg

Referenced by ColorText_has_arg().

#### 0.6.2.6.24 ColorArg_example()

```
char* ColorArg_example (
            ColorArg carg,
            bool colorized )
```

Create a string (`char*`) representation of a ColorArg with a stylized type/name using escape codes built from the ColorArg's values.

Parameters

| | | |
|---|---|---|
| `in` | *carg* | A ColorArg to get an example string for. |
| `in` | *colorized* | Whether to include a colorized example. If set to `false`, there will be no escape-codes in the string. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

ColorArg

#### 0.6.2.6.25 ColorArg_free()

```
void ColorArg_free (
            ColorArg * p )
```

Free allocated memory for a ColorArg.

This has no advantage over `free(colorarg)` right now, it is used in debugging, and may be extended in the future. It's better just to use it (or the colr_free() macro).

Parameters

| | | |
|---|---|---|
| in | *p* | ColorArg to free. |

See also

ColorArg

Referenced by _colr_free(), _colr_join(), ColorText_free_args(), colr_printf_handler(), colr_str_replace_all_ColorArg(), colr_str_replace_ColorArg(), colr_str_replace_re_all_ColorArg(), colr_str_replace_re_ColorArg(), colr_str_replace_re_match_ColorArg(), colr_str_replace_re_matches_ColorArg(), colr_str_replace_re_pat_all_ColorArg(), and colr_str_replace_re_pat_ColorArg().

#### 0.6.2.6.26 ColorArg_from_BasicValue()

```
ColorArg ColorArg_from_BasicValue (
            ArgType type,
            BasicValue value )
```

Explicit version of ColorArg_from_value that only handles BasicValues.

This is used in some macros to aid in dynamic escape code creation.

Parameters

| | | |
|---|---|---|
| in | *type* | ArgType (FORE, BACK, STYLE). |
| in | *value* | BasicValue to use. |

Returns

A ColorArg, with the `.value.type` member possibly set to `TYPE_INVALID`.

See also

ColorArg

#### 0.6.2.6.27 ColorArg_from_esc()

```
ColorArg ColorArg_from_esc (
            const char * s )
```

Parse an escape-code string (`char*`) into a ColorArg.

For malformed escape-codes the `.type` member will be `ARGTYPE_NONE`, and the `.value.type` member will be set to `TYPE_INVALID`. This means that `ColorArg_is_invalid(carg) == true`.

Parameters

| | | |
|---|---|---|
| in | *s* | The escape code to parse. It must not have extra characters. |

Returns

An initialized ColorArg, possibly invalid.

See also

ColorArg
colr_str_get_codes
ColorValue_from_esc
BasicValue_from_esc
ExtendedValue_from_esc
StyleValue_from_esc
RGB_from_esc

Referenced by ColorArgs_from_str().

#### 0.6.2.6.28 ColorArg_from_ExtendedValue()

```
ColorArg ColorArg_from_ExtendedValue (
            ArgType type,
            ExtendedValue value )
```

Explicit version of ColorArg_from_value that only handles ExtendedValues.

This is used in some macros to aid in dynamic escape code creation.

Parameters

| | | |
|---|---|---|
| in | *type* | ArgType (FORE, BACK, STYLE). |
| in | *value* | ExtendedValue to use. |

Returns

A ColorArg, with the `.value.type` member possibly set to `TYPE_INVALID`.

See also

ColorArg

#### 0.6.2.6.29 ColorArg_from_RGB()

```
ColorArg ColorArg_from_RGB (
            ArgType type,
            RGB value )
```

Explicit version of ColorArg_from_value that only handles RGB structs.

This is used in some macros to aid in dynamic escape code creation.

Parameters

| | | |
|---|---|---|
| in | *type* | ArgType (FORE, BACK, STYLE). |
| in | *value* | RGB struct to use. |

Returns

A ColorArg, with the `.value.type` member possibly set to `TYPE_INVALID`.

See also

ColorArg

#### 0.6.2.6.30 ColorArg_from_str()

```
ColorArg ColorArg_from_str (
            ArgType type,
            const char * colorname )
```

Build a ColorArg (fore, back, or style value) from a known color name/style.

The `.value.type` attribute can be checked for an invalid type, or you can call ColorArg_is_↩ invalid(x).

Parameters

| | | |
|---|---|---|
| in | *type* | ArgType (FORE, BACK, STYLE). |
| in | *colorname* | A known color name/style. |

Returns

A ColorArg struct with usable values.

See also

ColorArg

#### 0.6.2.6.31 ColorArg_from_StyleValue()

```
ColorArg ColorArg_from_StyleValue (
            ArgType type,
            StyleValue value )
```

Explicit version of ColorArg_from_value that only handles StyleValues.

This is used in some macros to aid in dynamic escape code creation.

Parameters

| | | |
|---|---|---|
| in | *type* | ArgType (FORE, BACK, STYLE). |
| in | *value* | StyleValue to use. |

Returns

A ColorArg, with the `.value.type` member possibly set to `TYPE_INVALID`.

See also

ColorArg

#### 0.6.2.6.32 ColorArg_from_value()

```
ColorArg ColorArg_from_value (
            ArgType type,
            ColorType colrtype,
            void * p )
```

Used with the color_arg macro to dynamically create a ColorArg based on it's argument type.

Parameters

| | | |
|---|---|---|
| in | *type* | ArgType value, to mark the type of ColorArg. |
| in | *colrtype* | ColorType value, to mark the type of ColorValue. |
| in | *p* | A pointer to either a BasicValue, ExtendedValue, or a RGB. |

Returns

A ColorArg struct with the appropriate `.value.type` member set for the value that was passed. For invalid types the `.value.type` member may be set to one of:

- TYPE_INVALID
- TYPE_INVALID_EXT_RANGE
- TYPE_INVALID_RGB_RANGE

See also

ColorArg

#### 0.6.2.6.33 ColorArg_is_empty()

```
bool ColorArg_is_empty (
            ColorArg carg )
```

Checks to see if a ColorArg is an empty placeholder.

A ColorArg is empty if it's `.type` is set to `ARGTYPE_NONE`.

Parameters

| | | |
|---|---|---|
| `in` | *carg* | A ColorArg to check. |

Returns

`true` if the ColorArg is considered "empty", otherwise `false`.

Referenced by ColorArg_length(), ColorArg_to_esc(), ColorArg_to_esc_s(), ColorText_has_args(), and ColorText_to_str().

#### 0.6.2.6.34 ColorArg_is_invalid()

```
bool ColorArg_is_invalid (
            ColorArg carg )
```

Checks to see if a ColorArg holds an invalid value.

Parameters

| | | |
|---|---|---|
| `in` | *carg* | ColorArg struct to check. |

Returns

`true` if the value is invalid, otherwise `false`.

See also

ColorArg

#### 0.6.2.6.35 ColorArg_is_ptr()

```
bool ColorArg_is_ptr (
            void * p )
```

Checks a void pointer to see if it contains a ColorArg struct.

The first member of a ColorArg is a marker.

Parameters

| | | |
|---|---|---|
| in | *p* | A void pointer to check. |

Returns

`true` if the pointer is a ColorArg, otherwise `false`.

See also

ColorArg

Referenced by _colr_free(), _colr_join(), _colr_join_array_length(), _colr_join_arrayn_size(), _colr_ptr_length(), _colr_ptr_repr(), _colr_ptr_to_str(), ColorText_from_values(), ColorText_set_values(), colr_join_arrayn(), and colr_printf_handler().

#### 0.6.2.6.36 ColorArg_is_valid()

```
bool ColorArg_is_valid (
            ColorArg carg )
```

Checks to see if a ColorArg holds a valid value.

Parameters

| | | |
|---|---|---|
| in | *carg* | ColorArg struct to check. |

Returns

`true` if the value is valid, otherwise `false`.

See also

ColorArg

#### 0.6.2.6.37 ColorArg_length()

```
size_t ColorArg_length (
            ColorArg carg )
```

Returns the length in bytes needed to allocate a string (`char*`) built with ColorArg_to_esc().

Parameters

| | | |
|---|---|---|
| in | *carg* | ColorArg to use. |

Returns

The length (`size_t`) needed to allocate a ColorArg's string, or `1` (size of an empty string) for invalid/empty arg types/values.

See also

ColorArg

Referenced by _colr_join_arrayn_size(), _colr_ptr_length(), and ColorText_length().

#### 0.6.2.6.38 ColorArg_repr()

```
char* ColorArg_repr (
            ColorArg carg )
```

Creates a string (`char*`) representation for a ColorArg.

Allocates memory for the string representation.

Parameters

| | | |
|---|---|---|
| in | *carg* | ColorArg struct to get the representation for. |

Returns

Allocated string for the representation.
*You must `free()` the memory allocated by this function.*

See also

ColorArg

Referenced by _colr_ptr_repr(), and ColorText_repr().

#### 0.6.2.6.39 ColorArg_to_esc()

```
char* ColorArg_to_esc (
            ColorArg carg )
```

Converts a ColorArg into an escape code string (`char*`).

Allocates memory for the string.

If the ColorArg is empty (`ARGTYPE_NONE`), an empty string is returned.

If the ColorValue is invalid, an empty string is returned. You must still free the empty string.

Parameters

| | | |
|---|---|---|
| `in` | *carg* | ColorArg to get the ArgType and ColorValue from. |

Returns

Allocated string for the escape code.
*You must `free()` the memory allocated by this function.* If the ColorArg is considered "empty", or the ColorValue is invalid, then `NULL` is returned.

See also

ColorArg

Referenced by _colr_join(), _colr_ptr_to_str(), ColorText_to_str(), colr_join_arrayn(), colr_printf_handler(), colr_str_replace_all_ColorArg(), colr_str_replace_ColorArg(), colr_str_replace_re_all_ColorArg(), colr_str_replace_re_ColorArg(), colr_str_replace_re_match_ColorArg(), colr_str_replace_re_matches_ColorArg(), colr_str_replace_re_pat_all_ColorArg(), and colr_str_replace_re_pat_ColorArg().

#### 0.6.2.6.40 ColorArg_to_esc_s()

```
bool ColorArg_to_esc_s (
            char * dest,
            ColorArg carg )
```

Converts a ColorArg into an escape code string (`char*`) and fills the destination string.

If the ColorArg is empty (`ARGTYPE_NONE`), `dest[0]` is set to "\0".

If the ColorValue is invalid, `dest[0]` is set to "\0".

Parameters

| | | |
|---|---|---|
| `in` | *dest* | Destination for the escape code string. *Must have room for the code type being used.* See ColorArg_length() for determining the size needed. |
| `in` | *carg* | ColorArg to get the ArgType and ColorValue from. |

Returns

`true` if the ColorArg was valid, otherwise `false`.

See also

ColorArg

#### 0.6.2.6.41 ColorArg_to_ptr()

```
ColorArg* ColorArg_to_ptr (
            ColorArg carg )
```

Copies a ColorArg into memory and returns the pointer.

You must free() the memory if you call this directly.

Parameters

| | | |
|---|---|---|
| `in` | *carg* | ColorArg to copy/allocate for. |

Returns

Pointer to a heap-allocated ColorArg.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

ColorArg

Referenced by ColorArgs_from_str().

#### 0.6.2.6.42 ColorArgs_array_free()

```
void ColorArgs_array_free (
            ColorArg ** ps )
```

Free an allocated array of ColorArgs, including the array itself.

Each individual ColorArg will be released, and finally the allocated memory for the array of pointers will be released.

Parameters

| | | |
|---|---|---|
| `in` | *ps* | A pointer to an array of ColorArgs, where `NULL` is the last item. |

#### 0.6.2.6.43 ColorArgs_array_repr()

```
char* ColorArgs_array_repr (
            ColorArg ** lst )
```

Creates a string representation for an array of ColorArg pointers.

Parameters

| | | |
|---|---|---|
| in | *lst* | The ColorArg array to create the representation for (`ColorArg**`). |

Returns

An allocated string, or `NULL` if `lst` is `NULL`, or the allocation fails.

#### 0.6.2.6.44 ColorArgs_from_str()

```
ColorArg** ColorArgs_from_str (
            const char * s,
            bool unique )
```

Create an array of ColorArgs from escape-codes found in a string (`char*`).

This uses ColorArg_from_esc() and colr_str_get_codes() to build a heap-allocated array of heap-allocated ColorArgs.

Parameters

| | | |
|---|---|---|
| in | *s* | A string to get the escape-codes from.<br>*Must be `null`-terminated.* |
| in | *unique* | Whether to only include *unique* ColorArgs. |

Returns

An allocated array of ColorArg pointers, where the last element is `NULL`.
*You must `free()` the memory allocated by this function.*

Return values

| | |
|---|---|
| *If* | `s` is `NULL`, or empty, or there are otherwise no escape-codes found in the string, then `NULL` is returned. |
| *On* | success, there will be at least two pointers behind the return value. The last pointer is always `NULL`. |

#### 0.6.2.6.45 ColorJustify_empty()

```
ColorJustify ColorJustify_empty (
            void  )
```

Creates an "empty" ColorJustify, with JUST_NONE set.

Returns

An initialized ColorJustify, with no justification method set.

See also

ColorJustify

Referenced by ColorText_empty().

#### 0.6.2.6.46 ColorJustify_eq()

```
bool ColorJustify_eq (
            ColorJustify a,
            ColorJustify b )
```

Compares two ColorJustify structs.

They are considered "equal" if their member values match.

Parameters

| | | |
|---|---|---|
| in | *a* | First ColorJustify to compare. |
| in | *b* | Second ColorJustify to compare. |

Returns

`true` if they are equal, otherwise `false`.

See also

ColorJustify

#### 0.6.2.6.47 ColorJustify_is_empty()

```
bool ColorJustify_is_empty (
            ColorJustify cjust )
```

Checks to see if a ColorJustify is "empty".

A ColorJustify is considered "empty" if the `.method` member is set to `JUST_NONE`.

Parameters

| | | |
|---|---|---|
| in | *cjust* | The ColorJustify to check. |

Returns

`true` if the ColorJustify is empty, otherwise `false`.

See also

ColorJustify
ColorJustify_empty

Referenced by ColorText_is_empty(), and ColorText_length().

#### 0.6.2.6.48 ColorJustify_new()

```
ColorJustify ColorJustify_new (
             ColorJustifyMethod method,
             int width,
             char padchar )
```

Creates a ColorJustify.

This is used to ensure every ColorJustify has it's `.marker` member set correctly.

Parameters

| | | |
|---|---|---|
| in | *method* | ColorJustifyMethod to use. |
| in | *width* | Width for justification. If `0` is given, ColorText will use the width from colr_term_size(). |
| in | *padchar* | Padding character to use. If `0` is given, the default, space (" "), is used. |

Returns

An initialized ColorJustify.

#### 0.6.2.6.49 ColorJustify_repr()

```
char* ColorJustify_repr (
             ColorJustify cjust )
```

Creates a string (`char*`) representation for a ColorJustify.

Allocates memory for the string representation.

Parameters

| | | |
|---|---|---|
| in | *cjust* | ColorJustify struct to get the representation for. |

Returns

Allocated string for the representation.
*You must `free()` the memory allocated by this function.*

See also

ColorJustify

Referenced by ColorText_repr().

#### 0.6.2.6.50 ColorJustifyMethod_repr()

```
char* ColorJustifyMethod_repr (
            ColorJustifyMethod meth )
```

Creates a string (`char*`) representation for a ColorJustifyMethod.

Allocates memory for the string representation.

Parameters

| | | |
|---|---|---|
| in | *meth* | ColorJustifyMethod to get the representation for. |

Returns

Allocated string for the representation.
*You must `free()` the memory allocated by this function.*

See also

ColorJustifyMethod

Referenced by ColorJustify_repr().

#### 0.6.2.6.51 ColorResult_empty()

```
ColorResult ColorResult_empty (
            void  )
```

Creates a ColorResult with `.result=NULL` and `.length=-1`, with the appropriate struct marker.

Returns

An "empty" (initialized) ColorResult.

See also

ColorResult

Referenced by ColorResult_new().

#### 0.6.2.6.52 ColorResult_eq()

```
bool ColorResult_eq (
            ColorResult a,
            ColorResult b )
```

Compares two ColorResults.

They are equal if all of their members are equal, excluding the memory address for the `.result` member.

Parameters

| | | |
|---|---|---|
| in | *a* | First ColorResult to compare. |
| in | *b* | Second ColorResult to compare. |

Returns

`true` if they are equal, otherwise `false`.

See also

ColorResult

#### 0.6.2.6.53 ColorResult_free()

```
void ColorResult_free (
            ColorResult * p )
```

Free allocated memory for a ColorResult and it's `.result` member.

Parameters

| | | |
|---|---|---|
| in | *p* | A ColorResult with a `NULL` or heap-allocated `.result` member. |

See also

ColorResult

Referenced by _colr_free(), _colr_join(), colr_printf_handler(), colr_str_replace_all_ColorResult(), colr_str_replace_ColorResult(), colr_str_replace_re_all_ColorResult(), colr_str_replace_re_ColorResult(), colr_str_replace_re_match_ColorResult(), colr_str_replace_re_matches_ColorResult(), colr_str_replace_re_pat_all_ColorResult(), and colr_str_replace_re_pat_ColorResult().

#### 0.6.2.6.54 ColorResult_is_ptr()

```
bool ColorResult_is_ptr (
            void * p )
```

Checks a void pointer to see if it contains a ColorResult struct.

The first member of a ColorResult is a marker.

Parameters

| | | |
|---|---|---|
| in | *p* | A void pointer to check. |

Returns

`true` if the pointer is a ColorResult, otherwise `false`.

See also

ColorResult

Referenced by _colr_free(), _colr_join(), _colr_join_array_length(), _colr_join_arrayn_size(), _colr_ptr_length(), _colr_ptr_repr(), _colr_ptr_to_str(), colr_join_arrayn(), and colr_printf_handler().

#### 0.6.2.6.55 ColorResult_length()

```
size_t ColorResult_length (
            ColorResult cres )
```

Return the length in bytes (including the null-terminator), that is needed to store the return from ColorResult_to_str() (`.result`).

Parameters

| | | |
|---|---|---|
| in | *cres* | A ColorResult to calculate the length for. |

Returns

The length of a ColorResult, possibly `0` if `.result` is `NULL`.

See also

ColorResult

Referenced by _colr_join_arrayn_size(), and _colr_ptr_length().

#### 0.6.2.6.56 ColorResult_new()

```
ColorResult ColorResult_new (
            char * s )
```

Initialize a new ColorResult with an allocated string (`char*`).

Parameters

| | | |
|---|---|---|
| `in` | *s* | An allocated string to use for the `.result` member. |

Returns

An initialized ColorResult.

See also

ColorResult

#### 0.6.2.6.57 ColorResult_repr()

```
char* ColorResult_repr (
            ColorResult cres )
```

Create a string representation for a ColorResult.

This happens to be the same as `colr_str_repr(cres.result)` right now.

Parameters

| | | |
|---|---|---|
| `in` | *cres* | A ColorResult to create the representation string for. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

ColorResult

Referenced by _colr_ptr_repr().

#### 0.6.2.6.58 ColorResult_to_ptr()

```
ColorResult* ColorResult_to_ptr (
            ColorResult cres )
```

Allocate memory for a ColorResult, fill it, and return it.

This ensure the appropriate struct marker is set, for use with Colr.

Parameters

| | | |
|---|---|---|
| in | *cres* | A ColorResult to use. |

Returns

An allocated ColorResult.
*You must `free()` the memory allocated by this function.*
If used inside of the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros, *they will `free()` the result*. Otherwise, *you are responsible for calling `free()`*.
*If allocation fails, `NULL` is returned*.

See also

ColorResult

#### 0.6.2.6.59 ColorResult_to_str()

```
char* ColorResult_to_str (
            ColorResult cres )
```

Convert a ColorResult into a string (`char*`).

This simply returns the `.result` member right now. It is used for compatibility with the colr_to_str() macro.

Parameters

| | | |
|---|---|---|
| in | *cres* | A ColorResult to use. |

Returns

A stringified-version if this ColorResult, which happens to be the `.result` member. *If you free the result of this function, the original string used to create the ColorResult will be lost.*

See also

ColorResult

Referenced by _colr_join(), _colr_ptr_to_str(), colr_join_arrayn(), colr_printf_handler(), colr_str_replace_all_ColorResult(), colr_str_replace_ColorResult(), colr_str_replace_re_all_ColorResult(), colr_str_replace_re_ColorResult(), colr_str_replace_re_match_ColorResult(), colr_str_replace_re_matches_ColorResult(), colr_str_replace_re_pat_all_ColorResult(), and colr_str_replace_re_pat_ColorResult().

#### 0.6.2.6.60 ColorText_empty()

```
ColorText ColorText_empty (
            void  )
```

Creates an "empty" ColorText with pointers set to `NULL`.

Returns

An initialized ColorText.

See also

ColorText

Referenced by ColorText_from_values(), and ColorText_set_values().

#### 0.6.2.6.61 ColorText_free()

```
void ColorText_free (
            ColorText * p )
```

Frees a ColorText and it's ColorArgs.

The text member is left alone, because it wasn't created by ColrC.

Parameters

| | | |
|---|---|---|
| in | *p* | Pointer to ColorText to free, along with it's Colr-based members. |

See also

ColorText

Referenced by _colr_free(), _colr_join(), colr_printf_handler(), colr_str_replace_all_ColorText(), colr_str_replace_ColorText(), colr_str_replace_re_all_ColorText(), colr_str_replace_re_ColorText(), colr_str_replace_re_match_ColorText(), colr_str_replace_re_matches_ColorText(), colr_str_replace_re_pat_all_ColorText(), and colr_str_replace_re_pat_ColorText().

#### 0.6.2.6.62 ColorText_free_args()

```
void ColorText_free_args (
            ColorText * p )
```

Frees the ColorArg members of a ColorText.

The ColorText itself is not free'd.

This is safe to use on a stack-allocated ColorText with heap-allocated ColorArgs.

Parameters

| | | |
|---|---|---|
| in | *p* | Pointer to a ColorText. |

See also

ColorText

Referenced by ColorText_free().

#### 0.6.2.6.63 ColorText_from_values()

```
ColorText ColorText_from_values (
            char * text,
             ...  )
```

Builds a ColorText from 1 mandatory string (`char*`), and optional fore, back, and style args (pointers to ColorArgs).

Parameters

| | | |
|---|---|---|
| in | *text* | Text to colorize (a regular string). |
| in | ... | ColorArgs for fore, back, and style, in any order. |

Returns

An initialized ColorText struct.

See also

ColorText

#### 0.6.2.6.64 ColorText_has_arg()

```
bool ColorText_has_arg (
            ColorText ctext,
            ColorArg carg )
```

Checks to see if a ColorText has a certain ColorArg value set.

Uses ColorArg_eq() to inspect the `fore`, `back`, and `style` members.

Parameters

| | | |
|---|---|---|
| `in` | *ctext* | The ColorText to inspect. |
| `in` | *carg* | The ColorArg to look for. |

Returns

`true` if the `fore`, `back`, or `style` arg matches `carg`, otherwise `false`.

See also

ColorText

#### 0.6.2.6.65 ColorText_has_args()

```
bool ColorText_has_args (
            ColorText ctext )
```

Checks to see if a ColorText has any argument values set.

Parameters

| | | |
|---|---|---|
| `in` | *ctext* | A ColorText to check. |

Returns

`true` if `.fore`, `.back`, or `.style` is set to a non-empty ColorArg, otherwise `false`.

See also

ColorText

#### 0.6.2.6.66 ColorText_is_empty()

```
bool ColorText_is_empty (
            ColorText ctext )
```

Checks to see if a ColorText has no usable values.

A ColorText is considered "empty" if the `.text`, `.fore`, `.back`, and `.style` pointers are `NULL`, and the `.just` member is set to an "empty" ColorJustify.

Parameters

| | | |
|---|---|---|
| `in` | *ctext* | The ColorText to check. |

Returns

`true` if the ColorText is empty, otherwise `false`.

See also

ColorText
ColorText_empty

#### 0.6.2.6.67 ColorText_is_ptr()

```
bool ColorText_is_ptr (
            void * p )
```

Checks a void pointer to see if it contains a ColorText struct.

The first member of a ColorText is a marker.

Parameters

| | | |
|---|---|---|
| `in` | *p* | A void pointer to check. |

Returns

`true` if the pointer is a ColorText, otherwise `false`.

See also

ColorText

Referenced by _colr_free(), _colr_join(), _colr_join_array_length(), _colr_join_arrayn_size(), _colr_↩ptr_length(), _colr_ptr_repr(), _colr_ptr_to_str(), colr_join_arrayn(), and colr_printf_handler().

#### 0.6.2.6.68 ColorText_length()

```
size_t ColorText_length (
            ColorText ctext )
```

Returns the length in bytes needed to allocate a string (`char*`) built with ColorText_to_str() with the current `text`, `fore`, `back`, and `style` members.

Parameters

| | | |
|---|---|---|
| `in` | *ctext* | ColorText to use. |

Returns

The length (`size_t`) needed to allocate a ColorText's string, or `1` (size of an empty string) for invalid/empty arg types/values.

See also

ColorText

Referenced by _colr_join_arrayn_size(), _colr_ptr_length(), and ColorText_to_str().

#### 0.6.2.6.69 ColorText_repr()

```
char* ColorText_repr (
            ColorText ctext )
```

Allocate a string (`char*`) representation for a ColorText.

Parameters

| | | |
|---|---|---|
| `in` | *ctext* | ColorText to get the string representation for. |

Returns

Allocated string for the ColorText.

See also

ColorText

Referenced by _colr_ptr_repr().

#### 0.6.2.6.70 ColorText_set_just()

```
ColorText* ColorText_set_just (
            ColorText * ctext,
            ColorJustify cjust )
```

Set the ColorJustify method for a ColorText, and return the ColorText.

This is to facilitate the justification macros. If you already have a pointer to a ColorText, you can just do `ctext->just = just;`. The purpose of this is to allow `ColorText_set_just(ColorText_to_ptr(...), ...)` to work.

Parameters

| | | |
|---|---|---|
| out | *ctext* | The ColorText to set the justification method for. |
| in | *cjust* | The ColorJustify struct to use. |

Returns

The same pointer that was given as `ctext`.

See also

ColorText

#### 0.6.2.6.71 ColorText_set_values()

```
void ColorText_set_values (
            ColorText * ctext,
            char * text,
             ...  )
```

Initializes an existing ColorText from 1 mandatory string (`char*`), and optional fore, back, and style args (pointers to ColorArgs).

Parameters

| | | |
|---|---|---|
| out | *ctext* | A ColorText to initialize with values. |
| in | *text* | Text to colorize (a regular string). |
| in | ... | A `va_list` with ColorArgs pointers for fore, back, and style, in any order. |

Returns

An initialized ColorText struct.

See also

ColorText

#### 0.6.2.6.72 ColorText_to_ptr()

```
ColorText* ColorText_to_ptr (
            ColorText ctext )
```

Copies a ColorText into allocated memory and returns the pointer.

You must free() the memory if you call this directly.

Parameters

| | | |
|---|---|---|
| in | *ctext* | ColorText to copy/allocate for. |

Returns

Pointer to a heap-allocated ColorText.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

ColorText

#### 0.6.2.6.73 ColorText_to_str()

```
char* ColorText_to_str (
            ColorText ctext )
```

Stringifies a ColorText struct, creating a mix of escape codes and text.

Parameters

| | | |
|---|---|---|
| in | *ctext* | ColorText to stringify. |

Returns

An allocated string with text/escape-codes.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*. If the ColorText has a `NULL .text` member, `NULL` is returned.

See also

ColorText

Referenced by _colr_join(), _colr_ptr_to_str(), colr_join_arrayn(), colr_printf_handler(), colr_str_replace_all_ColorText(), colr_str_replace_ColorText(), colr_str_replace_re_all_ColorText(), colr_str_replace_re_ColorText(), colr_str_replace_re_match_ColorText(), colr_str_replace_re_matches_ColorText(), colr_str_replace_re_pat_all_ColorText(), and colr_str_replace_re_pat_ColorText().

#### 0.6.2.6.74 ColorType_eq()

```
bool ColorType_eq (
            ColorType a,
            ColorType b )
```

Compares two ColorTypes.

This is used to implement colr_eq().

Parameters

| | | |
|---|---|---|
| `in` | *a* | The first ColorType to compare. |
| `in` | *b* | The second ColorType to compare. |

Returns

`true` if they are equal, otherwise `false`.

See also

ColorType

#### 0.6.2.6.75 ColorType_from_str()

```
ColorType ColorType_from_str (
            const char * arg )
```

Determine which type of color value is desired by name.

Example:

- "red" == TYPE_BASIC
- "253" == TYPE_EXTENDED
- "123,55,67" == TYPE_RGB

Parameters

| | | |
|---|---|---|
| in | *arg* | Color name to get the ColorType for. |

Return values

| | |
|---|---|
| *ColorType* | value on success. |
| *TYPE_INVALID* | for invalid color names/strings. |
| *TYPE_INVALID_EXT_RANGE* | for ExtendedValues outside of 0-255. |
| *TYPE_INVALID_RGB_RANGE* | for rgb values outside of 0-255. |

See also

ColorType

#### 0.6.2.6.76 ColorType_is_invalid()

```
bool ColorType_is_invalid (
            ColorType type )
```

Check to see if a ColorType value is considered invalid.

Parameters

| | | |
|---|---|---|
| in | *type* | ColorType value to check. |

Returns

`true` if the value is considered invalid, otherwise `false`.

See also

ColorType

#### 0.6.2.6.77 ColorType_is_valid()

```
bool ColorType_is_valid (
            ColorType type )
```

Check to see if a ColorType value is considered valid.

Parameters

| | | |
|---|---|---|
| in | *type* | ColorType value to check. |

Returns

`true` if the value is considered valid, otherwise `false`.

See also

ColorType

#### 0.6.2.6.78 ColorType_repr()

```
char* ColorType_repr (
            ColorType type )
```

Creates a string (`char*`) representation of a ColorType.

Parameters

| | | |
|---|---|---|
| in | *type* | A ColorType to get the type from. |

Returns

A pointer to an allocated string.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

ColorType

#### 0.6.2.6.79 ColorType_to_str()

```
char* ColorType_to_str (
            ColorType type )
```

Create a human-friendly string (`char*`) representation for a ColorType.

Parameters

| | | |
|---|---|---|
| in | *type* | A ColorType to get the name for. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

ColorType

Referenced by ColorValue_example().

#### 0.6.2.6.80 ColorValue_empty()

```
ColorValue ColorValue_empty (
            void  )
```

Create an "empty" ColorValue.

This is used with ColorArg_empty() to build ColorArgs that don't do anything, where using `NULL` has a different meaning inside the colr_cat(), colr_join(), Colr(), Colr_cat(), and Colr_join() macros.

Returns

`(ColorValue){.type=TYPE_NONE, .basic=0, .ext=0, .rgb=(RGB){0, 0, 0}}`

See also

ColorArg
ColorArg_empty
ColorArg_is_empty
ColorValue_is_empty

#### 0.6.2.6.81 ColorValue_eq()

```
bool ColorValue_eq (
            ColorValue a,
            ColorValue b )
```

Compares two ColorValue structs.

They are considered "equal" if all of their members match.

Parameters

| | | |
|---|---|---|
| in | *a* | First ColorValue to compare. |
| in | *b* | Second ColorValue to compare. |

Returns

`true` if they are equal, otherwise `false`.

See also

ColorValue

Referenced by ColorArg_eq().

#### 0.6.2.6.82 ColorValue_example()

```
char* ColorValue_example (
            ColorValue cval )
```

Create a string (`char*`) representation of a ColorValue with a human-friendly type/name.

Parameters

| | | |
|---|---|---|
| `in` | *cval* | A ColorValue to get an example string for. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

ColorValue

Referenced by ColorArg_example().

#### 0.6.2.6.83 ColorValue_from_esc()

```
ColorValue ColorValue_from_esc (
            const char * s )
```

Convert an escape-code string (`char*`) into a ColorValue.

Parameters

| | | |
|---|---|---|
| `in` | *s* | An escape-code string to parse.<br>*Must be `null`-terminated.* |

Returns

A ColorValue (with no fore/back information, only the color type and value).

Return values

| | |
|---|---|
| *For* | invalid strings, the `.type` member can be one of:<br>• TYPE_INVALID<br>• TYPE_INVALID_EXT_RANGE<br>• TYPE_INVALID_RGB_RANGE |

See also

ColorValue
ColorArg_from_esc

Referenced by ColorArg_from_esc().

#### 0.6.2.6.84 ColorValue_from_str()

```
ColorValue ColorValue_from_str (
            const char * s )
```

Create a ColorValue from a known color name, or RGB string (`char*`).

Parameters

| | | |
|---|---|---|
| `in` | *s* | A string to parse the color name from (can be an RGB string). |

Returns

A ColorValue (with no fore/back information, only the color type and value).

Return values

| | |
|---|---|
| *For* | invalid strings, the `.type` member can be one of:<br>• TYPE_INVALID<br>• TYPE_INVALID_EXT_RANGE<br>• TYPE_INVALID_RGB_RANGE |

See also

ColorValue

Referenced by ColorArg_from_str().

#### 0.6.2.6.85 ColorValue_from_value()

```
ColorValue ColorValue_from_value (
            ColorType type,
            void * p )
```

Used with the color_val macro to dynamically create a ColorValue based on it's argument type.

Parameters

| | | |
|---|---|---|
| in | *type* | A ColorType value, to mark the type of ColorValue. |
| in | *p* | A pointer to either a BasicValue, ExtendedValue, or a RGB. |

Returns

A ColorValue struct with the appropriate `.type` member set for the value that was passed. For invalid types the `.type` member may be set to one of:

- TYPE_INVALID
- TYPE_INVALID_EXT_RANGE
- TYPE_INVALID_RGB_RANGE

See also

ColorValue

Referenced by ColorArg_from_BasicValue(), ColorArg_from_ExtendedValue(), ColorArg_from_RGB(), ColorArg_from_StyleValue(), ColorValue_from_esc(), and ColorValue_from_str().

#### 0.6.2.6.86 ColorValue_has_BasicValue()

```
bool ColorValue_has_BasicValue (
            ColorValue cval,
            BasicValue bval )
```

Checks to see if a ColorValue has a BasicValue set.

Parameters

| | | |
|---|---|---|
| in | *cval* | ColorValue to check. |
| in | *bval* | BasicValue to look for. |

Returns

`true` if the ColorValue has the exact BasicValue set.

See also

ColorValue

#### 0.6.2.6.87 ColorValue_has_ExtendedValue()

```
bool ColorValue_has_ExtendedValue (
            ColorValue cval,
            ExtendedValue eval )
```

Checks to see if a ColorValue has a ExtendedValue set.

Parameters

| | | |
|---|---|---|
| in | *cval* | ColorValue to check. |
| in | *eval* | ExtendedValue to look for. |

Returns

`true` if the ColorValue has the exact ExtendedValue set.

See also

ColorValue

#### 0.6.2.6.88 ColorValue_has_RGB()

```
bool ColorValue_has_RGB (
            ColorValue cval,
            RGB rgb )
```

Checks to see if a ColorValue has a RGB value set.

Parameters

| | | |
|---|---|---|
| in | *cval* | ColorValue to check. |
| in | *rgb* | RGB value to look for. |

Returns

`true` if the ColorValue has the exact RGB value set.

See also

ColorValue

#### 0.6.2.6.89 ColorValue_has_StyleValue()

```
bool ColorValue_has_StyleValue (
            ColorValue cval,
            StyleValue sval )
```

Checks to see if a ColorValue has a StyleValue set.

Parameters

| | | |
|---|---|---|
| in | *cval* | ColorValue to check. |
| in | *sval* | StyleValue to look for. |

Returns

`true` if the ColorValue has the exact StyleValue set.

See also

ColorValue

#### 0.6.2.6.90 ColorValue_is_empty()

```
bool ColorValue_is_empty (
            ColorValue cval )
```

Checks to see if a ColorValue is an empty placeholder.

Parameters

| | | |
|---|---|---|
| in | *cval* | ColorValue to check. |

Returns

`true` if the ColorValue is "empty", otherwise `false`.

See also

ColorValue
ColorValue_empty
ColorArg_empty
ColorArg_is_empty

#### 0.6.2.6.91 ColorValue_is_invalid()

```
bool ColorValue_is_invalid (
            ColorValue cval )
```

Checks to see if a ColorValue holds an invalid value.

Parameters

| | | |
|---|---|---|
| in | *cval* | ColorValue struct to check. |

Returns

`true` if the value is invalid, otherwise `false`.

See also

ColorValue

Referenced by ColorArg_from_esc().

#### 0.6.2.6.92 ColorValue_is_valid()

```
bool ColorValue_is_valid (
            ColorValue cval )
```

Checks to see if a ColorValue holds a valid value.

Parameters

| | | |
|---|---|---|
| in | *cval* | ColorValue struct to check. |

Returns

`true` if the value is valid, otherwise `false`.

See also

ColorValue

#### 0.6.2.6.93 ColorValue_length()

```
size_t ColorValue_length (
            ArgType type,
            ColorValue cval )
```

Returns the length in bytes needed to allocate a string (`char*`) built with ColorValue_to_esc() with the specified ArgType and ColorValue.

Parameters

| | | |
|---|---|---|
| in | *type* | ArgType (FORE, BACK, STYLE) |
| in | *cval* | ColorValue to use. |

Returns

The length (size_t) needed to allocate a ColorValue's string, or 1 (size of an empty string) for invalid/empty arg types/values.

See also

ColorValue

Referenced by ColorArg_length().

#### 0.6.2.6.94 ColorValue_repr()

```
char* ColorValue_repr (
            ColorValue cval )
```

Creates a string (char*) representation of a ColorValue.

Parameters

| | | |
|---|---|---|
| in | *cval* | A ColorValue to get the type and value from. |

Returns

A pointer to an allocated string.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned.*

See also

ColorValue

Referenced by ColorArg_repr().

#### 0.6.2.6.95 ColorValue_to_esc()

```
char* ColorValue_to_esc (
            ArgType type,
            ColorValue cval )
```

Converts a ColorValue into an escape code string (char*).

Parameters

| | | |
|---|---|---|
| in | *type* | ArgType (FORE, BACK, STYLE) to build the escape code for. |
| in | *cval* | ColorValue to get the color value from. |

Returns

An allocated string with the appropriate escape code. For invalid values, an empty string is returned.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

ColorValue

Referenced by ColorArg_to_esc().

#### 0.6.2.6.96 ColorValue_to_esc_s()

```
bool ColorValue_to_esc_s (
            char * dest,
            ArgType type,
            ColorValue cval )
```

Converts a ColorValue into an escape code string (`char*`) and fills the destination string.

For invalid ArgType/ColorValue combinations, `dest[0]` is set to "\0".

Parameters

| | | |
|---|---|---|
| out | *dest* | Destination string for the escape code string. *Must have room for the code type being used*. |
| in | *type* | ArgType (FORE, BACK, STYLE) to build the escape code for. |
| in | *cval* | ColorValue to get the color value from. |

Returns

`true` if a proper ArgType/ColorValue combination was used, otherwise `false`.

See also

ColorValue

Referenced by ColorArg_to_esc_s().

#### 0.6.2.6.97 colr_alloc_regmatch()

```
regmatch_t* colr_alloc_regmatch (
            regmatch_t match )
```

Allocates space for a regmatch_t, initializes it, and returns a pointer to it.

Parameters

| | | |
|---|---|---|
| in | *match* | A `regmatch_t` to allocate for and copy. |

Returns

An allocated copy of the `regmatch_t`.

Referenced by colr_re_matches().

#### 0.6.2.6.98 colr_append_reset()

```
void colr_append_reset (
            char * s )
```

Appends CODE_RESET_ALL to a string (`char*`), but makes sure to do it before any newlines.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to append to. *Must have extra room for CODE_RESET_ALL*. *Must be `null`-terminated*. |

Referenced by _colr_join(), _rainbow(), ColorText_to_str(), and colr_join_arrayn().

#### 0.6.2.6.99 colr_char_escape_char()

```
char colr_char_escape_char (
            const char c )
```

Returns the char needed to represent an escape sequence in C.

The following characters are supported:

| Escape Sequence | Description Representation |
|---|---|
| \' | single quote |
| \" | double quote |
| \? | question mark |
| \\ | backslash |
| \a | audible bell |

| Escape Sequence | Description Representation |
|---|---|
| \ b | backspace |
| \ f | form feed - new page |
| \ n | line feed - new line |
| \ r | carriage return |
| \ t | horizontal tab |
| \ v | vertical tab |

Parameters

| | | |
|---|---|---|
| in | *c* | The character to check. |

Returns

The letter, without a backslash, needed to create an escape sequence. If the char doesn't need an escape sequence, it is simply returned.

Referenced by colr_str_repr().

#### 0.6.2.6.100 colr_char_in_str()

```
bool colr_char_in_str (
            const char * s,
            const char c )
```

Determines if a character exists in the given string (`char*`).

Parameters

| | | |
|---|---|---|
| in | *c* | Character to search for. |
| in | *s* | String to check.<br>Input *must be `null`-terminated*. |

Returns

`true` if `c` is found in `s`, otherwise `false`.

Referenced by colr_str_chars_lcount(), and colr_str_lstrip_chars().

#### 0.6.2.6.101 colr_char_is_code_end()

```
bool colr_char_is_code_end (
            const char c )
```

Determines if a character is suitable for an escape code ending.

`m` is used as the last character in color codes, but other characters can be used for escape sequences (such as "\x1b[2A", cursor up). Actual escape code endings can be in the range (`char`) 64-126 (inclusive).

Since ColrC only deals with color codes and maybe some cursor/erase codes, this function tests if the character is either `A-Z` or `a-z`.

For more information, see: https://en.wikipedia.org/wiki/ANSI_escape_code

Parameters

| | | |
|---|---|---|
| in | *c* | Character to test. |

Returns

`true` if the character is a possible escape code ending, otherwise `false`.

Referenced by colr_str_code_count(), colr_str_code_len(), colr_str_get_codes(), colr_str_is_codes(), colr_str_noncode_len(), and colr_str_strip_codes().

#### 0.6.2.6.102 colr_char_repr()

```
char* colr_char_repr (
            char c )
```

Creates a string (`char*`) representation for a char.

Parameters

| | | |
|---|---|---|
| in | *c* | Value to create the representation for. |

Returns

An allocated string.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

Referenced by ColorJustify_repr().

#### 0.6.2.6.103 colr_char_should_escape()

```
bool colr_char_should_escape (
            const char c )
```

Determines if an ascii character has an escape sequence in C.

The following characters are supported:

| Escape Sequence | Description Representation |
|---:|---|
| \' | single quote |
| \" | double quote |
| \? | question mark |
| \\ | backslash |
| \a | audible bell |
| \b | backspace |
| \f | form feed - new page |
| \n | line feed - new line |
| \r | carriage return |
| \t | horizontal tab |
| \v | vertical tab |

Parameters

| | | |
|---|---|---|
| in | *c* | The character to check. |

Returns

`true` if the character needs an escape sequence, otherwise `false`.

Referenced by colr_str_repr().

#### 0.6.2.6.104 colr_check_marker()

```
bool colr_check_marker (
            uint32_t marker,
            void * p )
```

Checks an unsigned int against the individual bytes behind a pointer's value.

This helps to guard against overflows, because only a single byte is checked at a time. If any byte doesn't match the marker, `false` is immediately returned, instead of continuing past the pointer's bounds.

Parameters

| | | |
|---|---|---|
| in | *marker* | A colr marker, like COLORARG_MARKER, COLORTEXT_MARKER, etc. |
| in | *p* | A pointer to check, to see if it starts with the marker. |

Returns

`true` if all bytes match the marker, otherwise `false`.

See also

ColorArg_is_ptr
ColorText_is_ptr

Referenced by _colr_is_last_arg(), ColorArg_is_ptr(), ColorResult_is_ptr(), and ColorText_is_ptr().

#### 0.6.2.6.105 colr_empty_str()

```
char* colr_empty_str (
            void  )
```

Allocates an empty string (`char*`).

This is for keeping the interface simple, so the return values from color functions with invalid values can be consistent.

Returns

Pointer to an allocated string consisting of '\0'.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

Referenced by colr_str_center(), colr_str_ljust(), colr_str_replace_re_match(), colr_str_rjust(), and colr_str_strip_codes().

#### 0.6.2.6.106 colr_free_re_matches()

```
void colr_free_re_matches (
            regmatch_t ** matches )
```

Free an array of allocated `regmatch_t`, like the return from colr_re_matches().

Parameters

| | | |
|---|---|---|
| `out` | *matches* | A pointer to an array of `regmatch_t` pointers. |

Referenced by colr_str_replace_re_pat_all().

#### 0.6.2.6.107 colr_join_array()

```
char* colr_join_array (
            void * joinerp,
            void * ps )
```

Join an array of strings (`char*`), `ColorArg`s, or `ColorText`s by another string (`char*`), ColorArg, or ColorText.

Parameters

| | | |
|---|---|---|
| in | *joinerp* | The joiner (any ColorArg, ColorText, or string (char*)). |
| in | *ps* | An array of pointers to ColorArgs, ColorTexts, or strings (char*). The array must have NULL as the last item. |

Returns

An allocated string with the result.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr
colr_join
colr_join_arrayn

0.6.2.6.108 colr_join_arrayn()

```
char* colr_join_arrayn (
            void * joinerp,
            void * ps,
            size_t count )
```

Join an array of strings (char*), ColorArgs, or ColorTexts by another string (char*), ColorArg, or ColorText.

Parameters

| | | |
|---|---|---|
| in | *joinerp* | The joiner (any ColorArg, ColorText, or string (char*)). |
| in | *ps* | An array of pointers to ColorArgs, ColorTexts, or strings (char*). The array must have at least a length of count, unless a NULL element is placed at the end. |
| in | *count* | The total number of items in the array. |

Returns

An allocated string with the result.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*. If any parameter is NULL, NULL is returned.

See also

colr
colr_join

Referenced by colr_join_array().

#### 0.6.2.6.109 colr_mb_len()

```
size_t colr_mb_len (
            const char * s,
            size_t length )
```

Like `mbrlen`, except it will return the length of the next N (`length`) multibyte characters in bytes.

/details Unlike colr_str_mb_len(), which returns the number of multibyte characters, this function will return the number of bytes that make up the next number (`length`) of multibyte characters.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to check. |
| `in` | *length* | Number of multibyte characters to get the length for. |

Returns

The number of bytes parsed in `s` to get at least `length` multibyte characters.

Return values

| | |
|---|---|
| *0* | if `s` is `NULL`/empty, or `length` is `0`. |
| *(size_t)-1* | if an invalid multibyte sequence is found at the start of `s`. |

See also

colr_str_mb_len
colr_is_valid_mblen

Referenced by _rainbow().

#### 0.6.2.6.110 colr_printf_handler()

```
int colr_printf_handler (
            FILE * fp,
            const struct printf_info * info,
            const void *const * args )
```

Handles printing with printf for Colr objects.

This function matches the required `typedef` in `printf.h` (`printf_function`), for handling a custom printf format char with `register_printf_specifier`.

Attention

This feature uses a GNU extension, and is only available when COLR_GNU is defined. See the documentation for COLR_GNU.

Parameters

| | | |
|---|---|---|
| in | *fp* | FILE pointer for output. |
| in | *info* | Info from printf about how to format the argument. |
| in | *args* | Argument list (with only 1 argument), containing a ColorArg, ColorResult, ColorText, or string (char*) to format. |

Returns

The number of characters written.

Referenced by colr_printf_register().

#### 0.6.2.6.111 colr_printf_info()

```
int colr_printf_info (
            const struct printf_info * info,
            size_t n,
            int * argtypes,
            int * sz )
```

Handles the arg count/size for the Colr printf handler.

This function matches the required `typedef` in `printf.h` (`printf_arginfo_size_function`) for handling a custom printf format char with `register_printf_specifier`.

Attention

This feature uses a GNU extension, and is only available when COLR_GNU is defined. See the documentation for COLR_GNU.

Parameters

| | | |
|---|---|---|
| in | *info* | Info from printf about how to format the argument. |
| in | *n* | Number of arguments for the format char. |
| out | *argtypes* | Type of arguments being handled, from an `enum` defined in `printf`. Colr uses/sets one argument, a `PA_POINTER` type. |
| out | *sz* | Size of the arguments. Not used in Colr. |

Returns

The number of argument types set in `argtypes`.

Referenced by colr_printf_register().

#### 0.6.2.6.112 colr_printf_register()

```
void colr_printf_register (
            void  )
```

Registers COLR_FMT_CHAR to handle Colr objects in the printf-family functions.

This function only needs to be called once and `register_printf_specifier` is only called the first time this function is called.

Attention

This feature uses a GNU extension, and is only available when COLR_GNU is defined. See the documentation for COLR_GNU.

#### 0.6.2.6.113 colr_re_matches()

```
regmatch_t** colr_re_matches (
            const char * s,
            regex_t * repattern )
```

Returns all `regmatch_t` matches for regex pattern in a string (`char*`).

Parameters

| | | |
|---|---|---|
| in | *s* | The string to search. |
| in | *repattern* | The pattern to look for. |

Returns

A pointer to an allocated array of `regmatch_t*`, or `NULL` if `s` is `NULL` or `repattern` is `NULL`. The last member is always `NULL`.
*You must `free()` the memory allocated by this function.*

Referenced by colr_str_replace_re_pat_all().

#### 0.6.2.6.114 colr_set_locale()

```
bool colr_set_locale (
            void  )
```

Sets the locale to `(LC_ALL, "")` if it hasn't already been set.

This is used for functions dealing with multibyte strings.

Returns

`true` if the locale had to be set, `false` if it was already set.

Referenced by colr_mb_len(), and colr_str_mb_len().

#### 0.6.2.6.115 colr_str_array_contains()

```
bool colr_str_array_contains (
            char ** lst,
            const char * s )
```

Determine if a string (`char*`) is in an array of strings (`char**`, where the last element is `NULL`).

Parameters

| | | |
|---|---|---|
| `in` | *lst* | The string array to look in. |
| `in` | *s* | The string to look for. |

Returns

`true` if the string is found, otherwise `false`.

Return values

| | |
|---|---|
| <*tt*>*false*</*tt*> | if `lst` is `NULL` or `s` is `NULL`. |

Referenced by colr_str_get_codes().

#### 0.6.2.6.116 colr_str_array_free()

```
void colr_str_array_free (
            char ** ps )
```

Free an allocated array of strings, including the array itself.

Each individual string will be released, and finally the allocated memory for the array of pointers will be released.

Parameters

| | | |
|---|---|---|
| `in` | *ps* | A pointer to an array of strings. |

Referenced by ColorArgs_from_str().

#### 0.6.2.6.117 colr_str_center()

```
char* colr_str_center (
            const char * s,
            int width,
            const char padchar )
```

Center-justifies a string (`char*`), ignoring escape codes when measuring the width.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to justify.<br>Input *must be `null`-terminated*. |
| in | *width* | The overall width for the resulting string. If set to '0', the terminal width will be used from colr_term_size(). |
| in | *padchar* | The character to pad with. If '0', then '' is used. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_str_ljust
colr_str_rjust
colr_term_size

Referenced by colr_printf_handler().

#### 0.6.2.6.118 colr_str_char_count()

```
size_t colr_str_char_count (
            const char * s,
            const char c )
```

Counts the number of characters (`c`) that are found in a string (`char*`) (`s`).

Returns `0` if `s` is `NULL`, or `c` is ''\0''.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to examine.<br>*Must be `null`-terminated*. |
| in | *c* | The character to count.<br>*Must not be `0`*. |

Returns

The number of times `c` occurs in `s`.

Referenced by _rainbow().

#### 0.6.2.6.119 colr_str_char_lcount()

```
size_t colr_str_char_lcount (
            const char * s,
            const char c )
```

Counts the number of characters (`c`) that are found at the beginning of a string (`char*`) (`s`).

Returns `0` if `s` is `NULL`, `c` is "\0", or the string doesn't start with `c`.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to examine.<br>*Must be `null`-terminated*. |
| `in` | *c* | The character to count.<br>*Must not be `0`*. |

Returns

The number of times `c` occurs at the start of `s`.

Referenced by colr_str_lstrip_char().

#### 0.6.2.6.120 colr_str_chars_lcount()

```
size_t colr_str_chars_lcount (
            const char *restrict s,
            const char *restrict chars )
```

Counts the number of characters that are found at the beginning of a string (`char*`) (`s`), where the character can be any of `chars`.

Returns `0` if `s` is `NULL`/empty, `chars` is `NULL`/empty, or the string doesn't start with any of the characters in `chars`.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to examine.<br>*Must be `null`-terminated*. |
| `in` | *chars* | The characters to count, in any order.<br>*Must not be `0`*. |

Returns

The number of times a character in `chars` occurs at the start of `s`.

Referenced by colr_str_lstrip_chars().

#### 0.6.2.6.121 colr_str_code_count()

```
size_t colr_str_code_count (
            const char * s )
```

Return the number of escape-codes in a string (`char*`).

Parameters

| | | |
|---|---|---|
| in | *s* | A string to count the escape-codes for.<br>*Must be `null`-terminated*. |

Returns

The number of escape-codes, or `0` if `s` is `NULL`, or doesn't contain any escape-codes.

Referenced by colr_str_get_codes().

#### 0.6.2.6.122 colr_str_code_len()

```
size_t colr_str_code_len (
            const char * s )
```

Return the number of bytes that make up all the escape-codes in a string (`char*`).

Parameters

| | | |
|---|---|---|
| in | *s* | A string to count the code-chars for.<br>*Must be `null`-terminated*. |

Returns

The number of escape-code characters, or `0` if `s` is `NULL`, or doesn't contain any escape-codes.

#### 0.6.2.6.123 colr_str_copy()

```
char* colr_str_copy (
            char *restrict dest,
            const char *restrict src,
            size_t length )
```

Copies a string (`char*`) like strncpy, but ensures null-termination.

If `src` is `NULL`, or `dest` is `NULL`, `NULL` is returned.

If `src` does not contain a null-terminator, *this function will truncate at `length` characters*.

If `src` is an empty string, then `dest[0]` will be "\0" (an empty string).

A null-terminator is always appended to `dest`.

`src` and `dest` must not overlap.

Parameters

| | | |
|---|---|---|
| in | *dest* | Memory allocated for new string. *Must have room for* `strlen(src) + 1` *or* `length + 1`. |
| in | *src* | Source string to copy. |
| in | *length* | Maximum characters to copy. *This does not include the null-terminator*. Usually set to `strlen(dest)`. |

Returns

On success, a pointer to dest is returned.

#### 0.6.2.6.124 colr_str_ends_with()

```
bool colr_str_ends_with (
            const char *restrict s,
            const char *restrict suffix )
```

Determine if one string (`char*`) ends with another.

`str` and `suffix` must not overlap.

Parameters

| | | |
|---|---|---|
| in | *s* | String to check.<br>*Must be `null`-terminated.* |
| in | *suffix* | Suffix to check for.<br>*Must be `null`-terminated.* |

Returns

True if `str` ends with `suffix`.
False if either is NULL, or the string doesn't end with the suffix.

Referenced by colr_append_reset().

#### 0.6.2.6.125 colr_str_get_codes()

```
char** colr_str_get_codes (
            const char * s,
            bool unique )
```

Get an array of escape-codes from a string (`char*`).

This function copies the escape-code strings, and the pointers to the heap, if any escape-codes are found in the string.

colr_str_array_free() can be used to easily `free()` the result of this function.

Parameters

| | | |
|---|---|---|
| in | *s* | A string to get the escape-codes from.<br>*Must be `null`-terminated*. |
| in | *unique* | Whether to only include *unique* escape codes. |

Returns

An allocated array of string (`char*`) pointers, where the last element is `NULL`.
*You must `free()` the memory allocated by this function.*

Return values

| | |
|---|---|
| *If* | `s` is `NULL`, or empty, or there are otherwise no escape-codes found in the string, or allocation fails for the strings/array, then `NULL` is returned. |
| *On* | success, there will be at least two pointers behind the return value. The last pointer is always `NULL`. |

Referenced by ColorArgs_from_str().

#### 0.6.2.6.126 colr_str_has_codes()

```
bool colr_str_has_codes (
            const char * s )
```

Determines if a string (`char*`) has ANSI escape codes in it.

This will detect any ansi escape code, not just colors.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to check. Can be `NULL`.<br>Input *must be `null`-terminated*. |

Returns

`true` if the string has at least one escape code, otherwise `false`.

See also

colr_str_is_codes

#### 0.6.2.6.127 colr_str_hash()

```
ColrHash colr_str_hash (
            const char * s )
```

Hash a string using djb2.

This is only used for simple, short, string (`char*`) hashing. It is not designed for cryptography.

There are some notes about collision rates for this function here.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to hash.<br>*Must be `null`-terminated.* |

Returns

A ColrHash (`unsigned long`) value with the hash.

Return values

| | |
|---|---|
| *0* | if `s` is `NULL`. |
| *COLR_HASH_SEED* | if `s` is an empty string. |

Referenced by colr_str_array_contains().

#### 0.6.2.6.128 colr_str_is_all()

```
bool colr_str_is_all (
            const char * s,
            const char c )
```

Determines whether a string (`char*`) consists of only one character, possibly repeated.

Parameters

| | | |
|---|---|---|
| in | *s* | String to check. |
| in | *c* | Character to test for. Must not be `0`. |

Returns

`true` if `s` contains only the character `c`, otherwise `false`.

#### 0.6.2.6.129 colr_str_is_codes()

```
bool colr_str_is_codes (
            const char * s )
```

Determines if a string (`char*`) is composed entirely of escape codes.

Returns `false` if the string is `NULL`, or empty.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to check.<br>Input *must be `null`-terminated*. |

Returns

`true` if the string is escape-codes only, otherwise `false`.

See also

colr_str_has_codes

#### 0.6.2.6.130 colr_str_is_digits()

```
bool colr_str_is_digits (
            const char * s )
```

Determines whether all characters in a string (`char*`) are digits.

If `s` is NULL or an empty string (`""`), `false` is returned.

Parameters

| | | |
|---|---|---|
| `in` | *s* | String to check.<br>Input *must be `null`-terminated*. |

Returns

`true` if all characters are digits (0-9), otherwise `false`.

Referenced by ExtendedValue_from_str().

#### 0.6.2.6.131 colr_str_ljust()

```
char* colr_str_ljust (
            const char * s,
```

```
            int width,
            const char padchar )
```

Left-justifies a string (char*), ignoring escape codes when measuring the width.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to justify.<br>Input *must be null-terminated*. |
| in | *width* | The overall width for the resulting string. If set to '0', the terminal width will be used from colr_term_size(). |
| in | *padchar* | The character to pad with. If '0', then " " is used. |

Returns

An allocated string with the result, or NULL if s is NULL.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_str_center
colr_str_rjust
colr_term_size

Referenced by colr_printf_handler().

#### 0.6.2.6.132 colr_str_lower()

```
void colr_str_lower (
            char * s )
```

Converts a string (char*) into lower case in place.

Input *must be null-terminated*.

If s is NULL, nothing is done.

Parameters

| | | |
|---|---|---|
| in | *s* | The input string to convert to lower case. |

#### 0.6.2.6.133 colr_str_lstrip()

```
size_t colr_str_lstrip (
            char *restrict dest,
            const char *restrict s,
            size_t length,
            const char c )
```

Strip a leading character from a string (char*), filling another string (char*) with the result.

dest and s should not overlap.

Parameters

| | | |
|---|---|---|
| out | *dest* | Destination char array. Must have room for strlen(s) + 1. |
| in | *s* | String to strip the character from. |
| in | *length* | Length of s, the input string. |
| in | *c* | Character to strip. If set to 0, all whitespace characters will be used (' ', '\n', '\t', '\v', '\f', '\r'). |

Returns

The number of c characters removed. May return 0 if s is NULL/empty, dest is NULL.

Referenced by colr_str_lstrip_char(), and RGB_from_hex().

#### 0.6.2.6.134 colr_str_lstrip_char()

```
char* colr_str_lstrip_char (
            const char * s,
            const char c )
```

Strips a leading character from a string (char*), and allocates a new string with the result.

Parameters

| | | |
|---|---|---|
| in | *s* | String to strip the character from. |
| in | *c* | Character to strip. If set to 0, all whitespace characters will be used (' ', '\n', '\t'). |

Returns

An allocated string with the result. May return NULL if s is NULL/empty.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

#### 0.6.2.6.135 colr_str_lstrip_chars()

```
char* colr_str_lstrip_chars (
            const char *restrict s,
            const char *restrict chars )
```

Removes certain characters from the start of a string (char*) and allocates a new string with the result.

The order of the characters in chars does not matter. If any of them are found at the start of a string, they will be removed.

```
colr_str_lstrip_chars("aabbccTEST", "bca") == "TEST"
```

s and chars must not overlap.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to strip. s *Must be null-terminated*. |
| in | *chars* | A string of characters to remove. Each will be removed from the start of the string. chars *Must be null-terminated*. |

Returns

An allocated string with the result. May return NULL if s or chars is NULL.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

#### 0.6.2.6.136 colr_str_mb_len()

```
size_t colr_str_mb_len (
            const char * s )
```

Returns the number of characters in a string (char*), taking into account possibly multibyte characters.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to get the length of. |

Returns

The number of characters, single and multibyte, or 0 if s is NULL, empty, or has invalid multibyte sequences.

See also

colr_mb_len

Referenced by _rainbow().

#### 0.6.2.6.137 colr_str_noncode_len()

```
size_t colr_str_noncode_len (
            const char * s )
```

Returns the length of string (char*), ignoring escape codes and the the null-terminator.

Parameters

| | | |
|---|---|---|
| in | *s* | String to get the length for.<br>Input *must be `null`-terminated*. |

Returns

The length of the string, as if it didn't contain escape codes. For non-escape-code strings, this is like `strlen()`. For `NULL` or "empty" strings, `0` is returned.

See also

colr_str_strip_codes

Referenced by ColorText_length(), colr_str_center(), colr_str_ljust(), and colr_str_rjust().

#### 0.6.2.6.138 colr_str_replace()

```
char* colr_str_replace (
            const char *restrict s,
            const char *restrict target,
            const char *restrict repl )
```

Replaces the first substring found in a string (`char*`).

Using `NULL` as a replacement is like using an empty string (""), which removes the `target` string from `s`.

For a more dynamic version, see the colr_replace and colr_replace_re macros.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *target* | The string to replace. |
| in | *repl* | The string to replace with. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, or `target` is `NULL`/empty.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace_ColorArg(), colr_str_replace_ColorResult(), and colr_str_replace_↩ ColorText().

#### 0.6.2.6.139 colr_str_replace_all()

```
char* colr_str_replace_all (
            const char *restrict s,
            const char *restrict target,
            const char *restrict repl )
```

Replaces the first substring found in a string (`char*`).

Using `NULL` as a replacement is like using an empty string (""), which removes the `target` string from `s`.

For a more dynamic version, see the colr_replace and colr_replace_re macros.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *target* | The string to replace. |
| `in` | *repl* | The string to replace with. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, or `target` is `NULL`/empty.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace_all_ColorArg(), colr_str_replace_all_ColorResult(), and colr_str_↩ replace_all_ColorText().

#### 0.6.2.6.140 colr_str_replace_all_ColorArg()

```
char* colr_str_replace_all_ColorArg (
            const char *restrict s,
            const char *restrict target,
            ColorArg * repl )
```

Replace all substrings in a string (`char*`) with a ColorArg's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *target* | The string to replace. |
| `in` | *repl* | The ColorArg to produce escape-codes to replace with. ColorArg_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, or `target` is `NULL`/empty.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.2.6.141 colr_str_replace_all_ColorResult()

```
char* colr_str_replace_all_ColorResult (
            const char *restrict s,
            const char *restrict target,
            ColorResult * repl )
```

Replace all substrings in a string (`char*`) with a ColorResult's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *target* | The string to replace. |
| `in` | *repl* | The ColorResult to produce escape-codes to replace with. ColorResult_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, or `target` is `NULL`/empty.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.2.6.142 colr_str_replace_all_ColorText()

```
char* colr_str_replace_all_ColorText (
            const char *restrict s,
            const char *restrict target,
            ColorText * repl )
```

Replace all substrings in a string (`char*`) with a ColorText's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *target* | The string to replace. |
| in | *repl* | The ColorText to produce text/escape-codes to replace with. ColorText_free() is called after the replacement is done. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, or target is NULL/empty.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.2.6.143 colr_str_replace_cnt()

```
char* colr_str_replace_cnt (
            const char *restrict s,
            const char *restrict target,
            const char *restrict repl,
            int count )
```

Replaces one or more substrings in a string (char*).

Using NULL as a replacement is like using an empty string (""), which removes the target string from s.

For a more dynamic version, see the colr_replace and colr_replace_re macros.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *target* | The string to replace. |
| in | *repl* | The string to replace with. |
| in | *count* | Number of substrings to replace, or 0 to replace all substrings. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, or target is NULL/empty.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace(), and colr_str_replace_all().

#### 0.6.2.6.144 colr_str_replace_ColorArg()

```
char* colr_str_replace_ColorArg (
            const char *restrict s,
            const char *restrict target,
            ColorArg * repl )
```

Replace a substring in a string (`char*`) with a ColorArg's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *target* | The string to replace. |
| `in` | *repl* | The ColorArg to produce escape-codes to replace with. ColorArg_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, or `target` is `NULL`/empty.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.2.6.145 colr_str_replace_ColorResult()

```
char* colr_str_replace_ColorResult (
            const char *restrict s,
            const char *restrict target,
            ColorResult * repl )
```

Replace a substring in a string (`char*`) with a ColorResult's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *target* | The string to replace. |
| in | *repl* | The ColorResult to produce escape-codes to replace with. ColorResult_free() is called after the replacement is done. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, or target is NULL/empty.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

### 0.6.2.6.146 colr_str_replace_ColorText()

```
char* colr_str_replace_ColorText (
            const char *restrict s,
            const char *restrict target,
            ColorText * repl )
```

Replace a substring in a string (char*) with a ColorText's string result.

Using NULL as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *target* | The string to replace. |
| in | *repl* | The ColorText to produce text/escape-codes to replace with. ColorText_free() is called after the replacement is done. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, or target is NULL/empty.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.2.6.147 colr_str_replace_re()

```
char* colr_str_replace_re (
            const char *restrict s,
            const char *restrict pattern,
            const char *restrict repl,
            int re_flags )
```

Replaces a substring from a regex pattern string (`char*`) in a string (`char*`).

Using `NULL` as a replacement is like using an empty string (""), which removes the `target` string from `s`.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *pattern* | The regex match object to find text to replace. |
| `in` | *repl* | The string to replace with. |
| `in` | *re_flags* | Flags for `regcomp()`. `REG_EXTENDED` is always used, whether flags are provided or not. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `pattern` is `NULL`, or the regex pattern
doesn't compile/match.
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace_re_ColorArg(), colr_str_replace_re_ColorResult(), and colr_str_↩ replace_re_ColorText().

#### 0.6.2.6.148 colr_str_replace_re_all()

```
char* colr_str_replace_re_all (
            const char *restrict s,
            const char *restrict pattern,
            const char *restrict repl,
            int re_flags )
```

Replaces all substrings from a regex pattern string (`char*`) in a string (`char*`).

Using `NULL` as a replacement is like using an empty string (""), which removes the `target` string from `s`.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *pattern* | The regex match object to find text to replace. |
| in | *repl* | The string to replace with. |
| in | *re_flags* | Flags for `regcomp()`. `REG_EXTENDED` is always used, whether flags are provided or not. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `pattern` is `NULL`, or the regex pattern
doesn't compile/match.
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace_re_all_ColorArg(), colr_str_replace_re_all_ColorResult(), and colr_↩ str_replace_re_all_ColorText().

### 0.6.2.6.149 colr_str_replace_re_all_ColorArg()

```
char* colr_str_replace_re_all_ColorArg (
            const char *restrict s,
            const char *restrict pattern,
            ColorArg * repl,
            int re_flags )
```

Replace all substrings from a regex pattern string (`char*`) in a string (`char*`) with a ColorArg's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *pattern* | The regex pattern to compile. |
| in | *repl* | The ColorArg to produce escape-codes to replace with. ColorArg_free() is called after the replacement is done. |
| in | *re_flags* | Flags for `regcomp()`. `REG_EXTENDED` is always used, whether flags are provided or not. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `pattern` is `NULL`, or the regex pattern doesn't compile/match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.2.6.150 colr_str_replace_re_all_ColorResult()

```
char* colr_str_replace_re_all_ColorResult (
            const char *restrict s,
            const char *restrict pattern,
            ColorResult * repl,
            int re_flags )
```

Replace all substrings from a regex pattern string (`char*`) in a string (`char*`) with a ColorResult's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *pattern* | The regex match object to find text to replace. |
| `in` | *repl* | The ColorResult to produce escape-codes to replace with. ColorResult_free() is called after the replacement is done. |
| `in` | *re_flags* | Flags for `regcomp()`. `REG_EXTENDED` is always used, whether flags are provided or not. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `pattern` is `NULL`, or the regex pattern doesn't compile/match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.2.6.151 colr_str_replace_re_all_ColorText()

```
char* colr_str_replace_re_all_ColorText (
            const char *restrict s,
            const char *restrict pattern,
            ColorText * repl,
            int re_flags )
```

Replace all substrings from a regex pattern string (`char*`) in a string (`char*`) with a ColorText's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *pattern* | The regex match object to find text to replace. |
| `in` | *repl* | The ColorText to produce text/escape-codes to replace with. ColorText_free() is called after the replacement is done. |
| `in` | *re_flags* | Flags for `regcomp()`. `REG_EXTENDED` is always used, whether flags are provided or not. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `pattern` is `NULL`, or the regex pattern doesn't compile/match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

colr_replace
colr_replace_re

#### 0.6.2.6.152 colr_str_replace_re_ColorArg()

```
char* colr_str_replace_re_ColorArg (
            const char *restrict s,
            const char *restrict pattern,
            ColorArg * repl,
            int re_flags )
```

Replace substrings from a regex pattern string (`char*`) in a string (`char*`) with a ColorArg's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *pattern* | The regex pattern to compile. |
| `in` | *repl* | The ColorArg to produce escape-codes to replace with. ColorArg_free() is called after the replacement is done. |
| `in` | *re_flags* | Flags for `regcomp()`. `REG_EXTENDED` is always used, whether flags are provided or not. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `pattern` is `NULL`, or the regex pattern doesn't compile/match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.2.6.153 colr_str_replace_re_ColorResult()

```
char* colr_str_replace_re_ColorResult (
            const char *restrict s,
            const char *restrict pattern,
            ColorResult * repl,
            int re_flags )
```

Replace substrings from a regex pattern string (`char*`) in a string (`char*`) with a ColorResult's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *pattern* | The regex match object to find text to replace. |
| `in` | *repl* | The ColorResult to produce escape-codes to replace with. ColorResult_free() is called after the replacement is done. |
| `in` | *re_flags* | Flags for `regcomp()`. `REG_EXTENDED` is always used, whether flags are provided or not. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `pattern` is `NULL`, or the regex pattern doesn't compile/match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.2.6.154 colr_str_replace_re_ColorText()

```
char* colr_str_replace_re_ColorText (
            const char *restrict s,
            const char *restrict pattern,
            ColorText * repl,
            int re_flags )
```

Replace substrings from a regex pattern string (`char*`) in a string (`char*`) with a ColorText's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *pattern* | The regex match object to find text to replace. |
| `in` | *repl* | The ColorText to produce text/escape-codes to replace with. ColorText_free() is called after the replacement is done. |
| `in` | *re_flags* | Flags for `regcomp()`. `REG_EXTENDED` is always used, whether flags are provided or not. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `pattern` is `NULL`, or the regex pattern doesn't compile/match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

colr_replace
colr_replace_re

#### 0.6.2.6.155 colr_str_replace_re_match()

```
char* colr_str_replace_re_match (
            const char *restrict s,
            regmatch_t * match,
            const char *restrict repl )
```

Replaces substrings from a single regex match (`regmatch_t*`) in a string (`char*`).

Using `NULL` as a replacement is like using an empty string (""), which removes the `target` string from `s`.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *match* | The regex match object to find text to replace. |
| `in` | *repl* | The string to replace with. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, match is NULL, or the regex pattern doesn't match.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace_re_match_ColorArg(), colr_str_replace_re_match_ColorResult(), colr_str_replace_re_match_ColorText(), and colr_str_replace_re_pat().

### 0.6.2.6.156 colr_str_replace_re_match_ColorArg()

```
char* colr_str_replace_re_match_ColorArg (
            const char *restrict s,
            regmatch_t * match,
            ColorArg * repl )
```

Replace substrings from a regex match (regmatch_t*) in a string (char*) with a ColorArg's string result.

Using NULL as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *match* | The regex match object to find text to replace. |
| in | *repl* | The ColorArg to produce escape-codes to replace with. ColorArg_free() is called after the replacement is done. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, match is NULL, or the regex pattern doesn't match.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.2.6.157 colr_str_replace_re_match_ColorResult()

```
char* colr_str_replace_re_match_ColorResult (
            const char *restrict s,
            regmatch_t * match,
            ColorResult * repl )
```

Replace substrings from a regex match (`regmatch_t*`) in a string (`char*`) with a ColorResult's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *match* | The regex match object to find text to replace. |
| `in` | *repl* | The ColorResult to produce escape-codes to replace with. ColorResult_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `match` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

colr_replace
colr_replace_re

#### 0.6.2.6.158 colr_str_replace_re_match_ColorText()

```
char* colr_str_replace_re_match_ColorText (
            const char *restrict s,
            regmatch_t * match,
            ColorText * repl )
```

Replace substrings from a regex match (`regmatch_t*`) in a string (`char*`) with a ColorText's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *match* | The regex match object to find text to replace. |
| `in` | *repl* | The ColorText to produce text/escape-codes to replace with. ColorText_free() is called after the replacement is done. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, match is NULL, or the regex pattern doesn't match.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.2.6.159 colr_str_replace_re_match_i()

```
char* colr_str_replace_re_match_i (
            const char *restrict ref,
            char * target,
            regmatch_t * match,
            const char *restrict repl )
```

Replaces substrings from a regex match (regmatch_t*) in a string (char*).

This modifies target in place. It must have capacity for the result.

Using NULL as a replacement is like using an empty string (""), which removes the target string from s.

Parameters

| | | |
|---|---|---|
| in | *ref* | The string to use for offset references. Can be target. Set this to the source string if target has not been filled yet. If target has been filled, you may use target for both ref and target. |
| out | *target* | The string to modify. Must have room for the resulting string. |
| in | *match* | The regex match object to find text to replace. |
| in | *repl* | The string to replace with. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, match is NULL, or the regex pattern doesn't match.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace_re_matches().

#### 0.6.2.6.160 colr_str_replace_re_matches()

```
char* colr_str_replace_re_matches (
            const char *restrict s,
            regmatch_t ** matches,
            const char *restrict repl )
```

Replaces substrings from an array of regex match (`regmatch_t*`) in a string (`char*`).

Using `NULL` as a replacement is like using an empty string (""), which removes the `target` string from `s`.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *matches* | Regex match objects to find text to replace. The array must have `NULL` as the last member. |
| `in` | *repl* | The string to replace with. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `match` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace_re_matches_ColorArg(), colr_str_replace_re_matches_Color↩Result(), colr_str_replace_re_matches_ColorText(), and colr_str_replace_re_pat_all().

#### 0.6.2.6.161 colr_str_replace_re_matches_ColorArg()

```
char* colr_str_replace_re_matches_ColorArg (
            const char *restrict s,
            regmatch_t ** matches,
            ColorArg * repl )
```

Replace substrings from an array of regex matches (`regmatch_t**`) in a string (`char*`) with a ColorArg's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *matches* | The regex match objects to find text to replace. |
| `in` | *repl* | The ColorArg to produce escape-codes to replace with. ColorArg_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `match` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.2.6.162 colr_str_replace_re_matches_ColorResult()

```
char* colr_str_replace_re_matches_ColorResult (
            const char *restrict s,
            regmatch_t ** matches,
            ColorResult * repl )
```

Replace substrings from an array of regex matches (`regmatch_t**`) in a string (`char*`) with a ColorResult's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *matches* | The regex match objects to find text to replace. |
| `in` | *repl* | The ColorResult to produce escape-codes to replace with. ColorResult_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `match` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.2.6.163 colr_str_replace_re_matches_ColorText()

```
char* colr_str_replace_re_matches_ColorText (
            const char *restrict s,
```

```
            regmatch_t ** matches,
            ColorText * repl )
```

Replace substrings from an array of regex matches (`regmatch_t**`) in a string (`char*`) with a ColorText's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *matches* | The regex match objects to find text to replace. |
| `in` | *repl* | The ColorText to produce text/escape-codes to replace with. ColorText_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `match` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.2.6.164 colr_str_replace_re_pat()

```
char* colr_str_replace_re_pat (
            const char *restrict s,
            regex_t * repattern,
            const char *restrict repl )
```

Replaces regex patterns in a string (`char*`).

Using `NULL` as a replacement is like using an empty string (""), which removes the `target` string from `s`.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *repattern* | The regex pattern to match (`regex_t*`). |
| `in` | *repl* | The string to replace with. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, repattern is NULL, or the regex pattern doesn't match.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace_re(), colr_str_replace_re_pat_ColorArg(), colr_str_replace_re_pat_↩ ColorResult(), and colr_str_replace_re_pat_ColorText().

#### 0.6.2.6.165 colr_str_replace_re_pat_all()

```
char* colr_str_replace_re_pat_all (
            const char *restrict s,
            regex_t * repattern,
            const char *restrict repl )
```

Replaces all matches to a regex pattern in a string (char*).

Using NULL as a replacement is like using an empty string (""), which removes the target string from s.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *repattern* | The regex pattern to match (regex_t*). |
| in | *repl* | The string to replace with. |

Returns

An allocated string with the result, or NULL if s is NULL/empty, repattern is NULL, or the regex pattern doesn't match.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_replace
colr_replace_re

Referenced by colr_str_replace_re_all(), colr_str_replace_re_pat_all_ColorArg(), colr_str_replace_↩ re_pat_all_ColorResult(), and colr_str_replace_re_pat_all_ColorText().

#### 0.6.2.6.166 colr_str_replace_re_pat_all_ColorArg()

```
char* colr_str_replace_re_pat_all_ColorArg (
            const char *restrict s,
            regex_t * repattern,
            ColorArg * repl )
```

Replace all matches to a regex pattern in a string (`char*`) with a ColorArg's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *repattern* | The regex pattern to match (`regex_t*`). |
| `in` | *repl* | The ColorArg to produce escape-codes to replace with. ColorArg_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `repattern` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

colr_replace
colr_replace_re

#### 0.6.2.6.167 colr_str_replace_re_pat_all_ColorResult()

```
char* colr_str_replace_re_pat_all_ColorResult (
            const char *restrict s,
            regex_t * repattern,
            ColorResult * repl )
```

Replace all matches to a regex pattern in a string (`char*`) with a ColorResult's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *repattern* | The regex pattern to match (`regex_t*`). |
| `in` | *repl* | The ColorResult to produce escape-codes to replace with. ColorResult_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `repattern` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.2.6.168 colr_str_replace_re_pat_all_ColorText()

```
char* colr_str_replace_re_pat_all_ColorText (
            const char *restrict s,
            regex_t * repattern,
            ColorText * repl )
```

Replace all matches to a regex pattern in a string (`char*`) with a ColorText's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *repattern* | The regex pattern to match (`regex_t*`). |
| `in` | *repl* | The ColorText to produce text/escape-codes to replace with. ColorText_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `repattern` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.2.6.169 colr_str_replace_re_pat_ColorArg()

```
char* colr_str_replace_re_pat_ColorArg (
            const char *restrict s,
            regex_t * repattern,
            ColorArg * repl )
```

Replace regex patterns in a string (`char*`) with a ColorArg's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *repattern* | The regex pattern to match (`regex_t*`). |
| in | *repl* | The ColorArg to produce escape-codes to replace with. ColorArg_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `repattern` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

colr_replace
colr_replace_re

### 0.6.2.6.170 colr_str_replace_re_pat_ColorResult()

```
char* colr_str_replace_re_pat_ColorResult (
            const char *restrict s,
            regex_t * repattern,
            ColorResult * repl )
```

Replace regex patterns in a string (`char*`) with a ColorResult's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| in | *s* | The string to operate on. |
| in | *repattern* | The regex pattern to match (`regex_t*`). |
| in | *repl* | The ColorResult to produce escape-codes to replace with. ColorResult_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `repattern` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

colr_replace
colr_replace_re

#### 0.6.2.6.171 colr_str_replace_re_pat_ColorText()

```
char* colr_str_replace_re_pat_ColorText (
            const char *restrict s,
            regex_t * repattern,
            ColorText * repl )
```

Replace regex patterns in a string (`char*`) with a ColorText's string result.

Using `NULL` as a replacement is like using an empty string ("").

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to operate on. |
| `in` | *repattern* | The regex pattern to match (`regex_t*`). |
| `in` | *repl* | The ColorText to produce text/escape-codes to replace with. ColorText_free() is called after the replacement is done. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`/empty, `repattern` is `NULL`, or the regex pattern doesn't match.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_replace
colr_replace_re

#### 0.6.2.6.172 colr_str_repr()

```
char* colr_str_repr (
            const char * s )
```

Convert a string (`char*`) into a representation of a string, by wrapping it in quotes and escaping characters that need escaping.

If `s` is NULL, then an allocated string containing the string "NULL" is returned (without quotes).

Escape codes will be escaped, so the terminal will ignore them if the result is printed.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to represent. |

Returns

An allocated string with the representation.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_char_should_escape
colr_char_escape_char

Referenced by _colr_ptr_repr(), ColorResult_repr(), and ColorText_repr().

### 0.6.2.6.173 colr_str_rjust()

```
char* colr_str_rjust (
            const char * s,
            int width,
            const char padchar )
```

Right-justifies a string (`char*`), ignoring escape codes when measuring the width.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to justify.<br>Input *must be `null`-terminated*. |
| `in` | *width* | The overall width for the resulting string. If set to '0', the terminal width will be used from colr_term_size(). |
| `in` | *padchar* | The character to pad with. If '0', then " " is used. |

Returns

An allocated string with the result, or `NULL` if `s` is `NULL`.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

colr_str_center
colr_str_ljust
colr_term_size

Referenced by colr_printf_handler().

0.6.2.6.174 colr_str_starts_with()

```
bool colr_str_starts_with (
            const char *restrict s,
            const char *restrict prefix )
```

Checks a string (`char*`) for a certain prefix substring.

`prefix` *Must be `null`-terminated*.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to check. |
| in | *prefix* | The prefix string to look for. |

Returns

True if the string s starts with prefix.
False if one of the strings is null, or the prefix isn't found.

#### 0.6.2.6.175 colr_str_strip_codes()

```
char* colr_str_strip_codes (
            const char * s )
```

Strips escape codes from a string (char*), resulting in a new allocated string.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to strip escape codes from.<br>Input *must be null-terminated*. |

Returns

An allocated string with the result.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

See also

colr_str_noncode_len

Referenced by colr_printf_handler().

#### 0.6.2.6.176 colr_str_to_lower()

```
char* colr_str_to_lower (
            const char * s )
```

Allocate a new lowercase version of a string (char*).

*You must free() the memory allocated by this function.*

Parameters

| | | |
|---|---|---|
| in | *s* | The input string to convert to lower case.<br>*Must be `null`-terminated*. |

Returns

The allocated string, or `NULL` if `s` is `NULL`.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

Referenced by ExtendedValue_from_str(), and RGB_from_str().

#### 0.6.2.6.177 colr_supports_rgb()

```
bool colr_supports_rgb (
            void  )
```

Determine whether the current environment support RGB (True Colors).

This checks `$COLORTERM` for the appropriate value (`'truecolor'` or `'24bit'`). On "dumber" terminals, RGB codes are probably ignored or mistaken for a 256-color or even 8-color value.

For instance, RGB is supported in `konsole`, but not in `xterm` or `linux` ttys. Using RGB codes in `xterm` makes the colors appear as though a 256-color value was used (closest matching value, like RGB_to_term_RGB()). Using RGB codes in a simpler `linux` tty makes them appear as though an 8-color value was used. Very ugly, but not a disaster.

I haven't seen a *modern* linux terminal spew garbage across the screen from using RGB codes when they are not supported, but I could be wrong. I would like to see that terminal if you know of one.

Returns

`true` if 24-bit (true color, or "rgb") support is detected, otherwise `false`.

Referenced by colr_supports_rgb_static().

#### 0.6.2.6.178 colr_supports_rgb_static()

```
bool colr_supports_rgb_static (
            void  )
```

Same as colr_supports_rgb(), but the environment is only checked on the first call.

All other calls return the same result as the first call.

Returns

`true` if 24-bit (true color, or "rgb") support is detected, otherwise `false`.

#### 0.6.2.6.179 colr_term_size()

```
TermSize colr_term_size (
            void  )
```

Attempts to retrieve the row/column size of the terminal and returns a TermSize.

If the call fails, the environment variables `LINES` and `COLUMNS` are checked. If that fails, a default `TermSize` struct is returned:

```
(TermSize){.rows=35, .columns=80}
```

Returns

A TermSize struct with terminal size information.

Referenced by ColorText_length(), colr_str_center(), colr_str_ljust(), and colr_str_rjust().

#### 0.6.2.6.180 colr_win_size()

```
struct winsize colr_win_size (
            void  )
```

Attempts to retrieve a `winsize` struct from an `ioctl` call.

If the call fails, the environment variables `LINES` and `COLUMNS` are checked. If that fails, a default `winsize` struct is returned:

```
(struct winsize){.ws_row=35, .ws_col=80, .ws_xpixel=0, .ws_ypixel=0}
```

`man ioctl_tty` says that `.ws_xpixel` and `.ws_ypixel` are unused.

Returns

A `winsize` struct (`sys/ioctl.h`) with window size information.

Referenced by colr_term_size().

#### 0.6.2.6.181 colr_win_size_env()

```
struct winsize colr_win_size_env (
            void  )
```

Get window/terminal size using the environment variables `LINES`, `COLUMNS`, or `COLS`.

This is used as a fallback if the `ioctl()` call fails in colr_win_size(). If environment variables are not available, a default `winsize` struct is returned:

```
(struct winsize){.ws_row=35, .ws_col=80, .ws_xpixel=0, .ws_ypixel=0}
```

Returns

A `winsize` struct (`sys/ioctl.h`) with window size information.

Referenced by colr_win_size().

#### 0.6.2.6.182 ExtendedValue_eq()

```
bool ExtendedValue_eq (
            ExtendedValue a,
            ExtendedValue b )
```

Compares two ExtendedValues.

This is used to implement colr_eq().

Parameters

| | | |
|---|---|---|
| `in` | *a* | The first ExtendedValue to compare. |
| `in` | *b* | The second ExtendedValue to compare. |

Returns

`true` if they are equal, otherwise `false`.

See also

ExtendedValue

#### 0.6.2.6.183 ExtendedValue_from_BasicValue()

```
int ExtendedValue_from_BasicValue (
            BasicValue bval )
```

Convert a BasicValue into an ExtendedValue.

BASIC_INVALID, and other invalid BasicValues will return EXT_INVALID.

Parameters

| | | |
|---|---|---|
| in | *bval* | BasicValue to convert. |

Returns

An ExtendedValue `0-15` on success, otherwise EXT_INVALID.

See also

ExtendedValue

#### 0.6.2.6.184 ExtendedValue_from_esc()

```
int ExtendedValue_from_esc (
            const char * s )
```

Convert an escape-code string (`char*`) to an ExtendedValue.

Parameters

| | | |
|---|---|---|
| in | *s* | Escape-code string.<br>*Must be `null`-terminated.* |

Return values

| | |
|---|---|
| *An* | integer in the range `0-255` on success. |
| *EXT_INVALID* | on error (or if `s` is `NULL`). |
| *EXT_INVALID_RANGE* | if the code number was outside of the range `0-255`. |

See also

ExtendedValue

#### 0.6.2.6.185 ExtendedValue_from_hex()

```
int ExtendedValue_from_hex (
            const char * hexstr )
```

Create an ExtendedValue from a hex string (`char*`).

This is not a 1:1 translation of hex to rgb. Use RGB_from_hex() for that. This will convert the hex string to the closest matching ExtendedValue value.

The format for hex strings can be one of:

- "[#]ffffff" (Leading hash symbol is optional)
- "[#]fff" (short-form)

Parameters

| | | |
|---|---|---|
| in | *hexstr* | Hex string to convert. |

Returns

A value between 0 and 255 on success.

Return values

| | |
|---|---|
| *COLOR_INVALID* | on error or bad values. |

See also

ExtendedValue

Referenced by ExtendedValue_from_hex_default(), and ExtendedValue_from_str().

#### 0.6.2.6.186 ExtendedValue_from_hex_default()

```
ExtendedValue ExtendedValue_from_hex_default (
            const char * hexstr,
            ExtendedValue default_value )
```

Create an ExtendedValue from a hex string (`char*`), but return a default value if the hex string is invalid.

This is not a 1:1 translation of hex to rgb. Use RGB_from_hex_default() for that. This will convert the hex string to the closest matching ExtendedValue value.

The format for hex strings can be one of:

- "[#]ffffff" (Leading hash symbol is optional)
- "[#]fff" (short-form)

Parameters

| | | |
|---|---|---|
| in | *hexstr* | Hex string to convert. |
| in | *default_value* | ExtendedValue to use for bad hex strings. |

Returns

An ExtendedValue on success, or `default_value` on error.

See also

ExtendedValue
ExtendedValue_from_hex

#### 0.6.2.6.187 ExtendedValue_from_RGB()

```
ExtendedValue ExtendedValue_from_RGB (
            RGB rgb )
```

Convert an RGB value into the closest matching ExtendedValue.

Parameters

| | | |
|---|---|---|
| in | *rgb* | RGB value to convert. |

Returns

An ExtendedValue that closely matches the original RGB value.

See also

ExtendedValue

Referenced by ExtendedValue_from_hex(), format_bg_RGB_term(), and format_fg_RGB_term().

#### 0.6.2.6.188 ExtendedValue_from_str()

```
int ExtendedValue_from_str (
            const char * arg )
```

Converts a known name, integer string (0-255), or a hex string (`char*`), into an ExtendedValue suitable for the extended-value-based functions.

Hex strings can be used:

- "#ffffff" (Leading hash symbol is **NOT** optional)
- "#fff" (short-form)

The "#" is not optional for hex strings because it is impossible to tell the difference between the hex value '111' and the extended value '111' without it.

Parameters

| | | |
|---|---|---|
| in | *arg* | Color name to find the ExtendedValue for. |

Returns

A value between 0 and 255 on success.

Return values

| | |
|---|---|
| *EXT_INVALID* | on error or bad values. |
| *EXT_INVALID_RANGE* | if the number was outside of the range `0-255`. |

See also

ExtendedValue

#### 0.6.2.6.189 ExtendedValue_is_invalid()

```
bool ExtendedValue_is_invalid (
            int eval )
```

Determines whether an integer is an invalid ExtendedValue.

Parameters

| | | |
|---|---|---|
| `in` | *eval* | A number to check. |

Returns

`true` if the value is considered invalid, otherwise `false`.

See also

ExtendedValue

#### 0.6.2.6.190 ExtendedValue_is_valid()

```
bool ExtendedValue_is_valid (
            int eval )
```

Determines whether an integer is a valid ExtendedValue.

Parameters

| | | |
|---|---|---|
| `in` | *eval* | A number to check. |

Returns

`true` if the value is considered valid, otherwise `false`.

See also

ExtendedValue

#### 0.6.2.6.191 ExtendedValue_repr()

```
char* ExtendedValue_repr (
            int eval )
```

Creates a string (`char*`) representation of a ExtendedValue.

Parameters

| | | |
|---|---|---|
| `in` | *eval* | A ExtendedValue to get the value from. |

Returns

A pointer to an allocated string.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

ExtendedValue

#### 0.6.2.6.192 ExtendedValue_to_str()

```
char* ExtendedValue_to_str (
            ExtendedValue eval )
```

Creates a human-friendly string (`char*`) from an ExtendedValue's actual value, suitable for use with ExtendedValue_from_str().

Parameters

| | | |
|---|---|---|
| `in` | *eval* | A ExtendedValue to get the value from. |

Returns

A pointer to an allocated string
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

ExtendedValue

#### 0.6.2.6.193 format_bg()

```
void format_bg (
            char * out,
            BasicValue value )
```

Create an escape code for a background color.

Parameters

| | | |
|---|---|---|
| out | *out* | Memory allocated for the escape code string. *Must have enough room for CODEX_LEN*. |
| in | *value* | BasicValue value to use for background. |

#### 0.6.2.6.194 format_bg_RGB()

```
void format_bg_RGB (
            char * out,
            RGB rgb )
```

Create an escape code for a true color (rgb) background color using values from an RGB struct.

Parameters

| | | |
|---|---|---|
| out | *out* | Memory allocated for the escape code string. *Must have enough room for CODE_RGB_LEN*. |
| in | *rgb* | RGB struct to get red, blue, and green values from. |

Referenced by _rainbow(), and rainbow_bg().

#### 0.6.2.6.195 format_bg_RGB_term()

```
void format_bg_RGB_term (
            char * out,
            RGB rgb )
```

Create an escape code for a true color (rgb) fore color using an RGB struct's values, approximating 256-color values.

Parameters

| | | |
|---|---|---|
| out | *out* | Memory allocated for the escape code string. |
| in | *rgb* | Pointer to an RGB struct. |

Referenced by _rainbow(), and rainbow_bg_term().

#### 0.6.2.6.196 format_bgx()

```
void format_bgx (
            char * out,
            unsigned char num )
```

Create an escape code for an extended background color.

Parameters

| | | |
|---|---|---|
| out | *out* | Memory allocated for the escape code string. *Must have enough room for CODEX_LEN*. |
| in | *num* | Value to use for background. |

Referenced by format_bg_RGB_term().

#### 0.6.2.6.197 format_fg()

```
void format_fg (
            char * out,
            BasicValue value )
```

Create an escape code for a fore color.

Parameters

| | | |
|---|---|---|
| out | *out* | Memory allocated for the escape code string. *Must have enough room for CODEX_LEN*. |
| in | *value* | BasicValue value to use for fore. |

#### 0.6.2.6.198 format_fg_RGB()

```
void format_fg_RGB (
            char * out,
            RGB rgb )
```

Create an escape code for a true color (rgb) fore color using an RGB struct's values.

Parameters

| | | |
|---|---|---|
| out | *out* | Memory allocated for the escape code string. |
| in | *rgb* | Pointer to an RGB struct. |

Referenced by rainbow_fg().

#### 0.6.2.6.199 format_fg_RGB_term()

```
void format_fg_RGB_term (
            char * out,
            RGB rgb )
```

Create an escape code for a true color (rgb) fore color using an RGB struct's values, approximating 256-color values.

Parameters

| | | |
|---|---|---|
| out | *out* | Memory allocated for the escape code string. |
| in | *rgb* | Pointer to an RGB struct. |

Referenced by rainbow_fg_term().

#### 0.6.2.6.200 format_fgx()

```
void format_fgx (
            char * out,
            unsigned char num )
```

Create an escape code for an extended fore color.

Parameters

| | | |
|---|---|---|
| out | *out* | Memory allocated for the escape code string. *Must have enough room for CODEX_LEN*. |
| in | *num* | Value to use for fore. |

Referenced by format_fg_RGB_term().

#### 0.6.2.6.201 format_style()

```
void format_style (
            char * out,
            StyleValue style )
```

Create an escape code for a style.

Parameters

| | | |
|---|---|---|
| out | *out* | Memory allocated for the escape code string. *Must have enough room for STYLE_LEN*. |
| in | *style* | StyleValue value to use for style. |

#### 0.6.2.6.202 rainbow_bg()

```
char* rainbow_bg (
            const char * s,
            double freq,
            size_t offset,
            size_t spread )
```

Rainbow-ize some text using rgb back colors, lolcat style.

This prepends a color code to every character in the string. Multi-byte characters are handled properly by checking colr_mb_len(), and copying the bytes to the resulting string without codes between the multibyte characters.

The CODE_RESET_ALL code is appended to the result.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to colorize.<br>Input *must be null-terminated*. |
| in | *freq* | Frequency ("tightness") for the colors. |
| in | *offset* | Starting offset in the rainbow. |
| in | *spread* | Number of characters per color. |

Returns

The allocated/formatted string on success.
*You must free() the memory allocated by this function.*
*If allocation fails, NULL is returned*.

#### 0.6.2.6.203 rainbow_bg_term()

```
char* rainbow_bg_term (
            const char * s,
            double freq,
            size_t offset,
            size_t spread )
```

This is exactly like rainbow_bg(), except it uses colors that are closer to the standard 256-color values.

This prepends a color code to every character in the string. Multi-byte characters are handled properly by checking colr_mb_len(), and copying the bytes to the resulting string without codes between the multibyte characters.

The CODE_RESET_ALL code is appended to the result.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to colorize.<br>Input *must be `null`-terminated*. |
| in | *freq* | Frequency ("tightness") for the colors. |
| in | *offset* | Starting offset in the rainbow. |
| in | *spread* | Number of characters per color. |

Returns

The allocated/formatted string on success.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

#### 0.6.2.6.204 rainbow_fg()

```
char* rainbow_fg (
            const char * s,
            double freq,
            size_t offset,
            size_t spread )
```

Rainbow-ize some text using rgb fore colors, lolcat style.

This prepends a color code to every character in the string. Multi-byte characters are handled properly by checking colr_mb_len(), and copying the bytes to the resulting string without codes between the multibyte characters.

The `CODE_RESET_ALL` code is appended to the result.

Parameters

| | | |
|---|---|---|
| in | *s* | The string to colorize.<br>Input *must be `null`-terminated*. |
| in | *freq* | Frequency ("tightness") for the colors. |
| in | *offset* | Starting offset in the rainbow. |
| in | *spread* | Number of characters per color. |

Returns

The allocated/formatted string on success.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

#### 0.6.2.6.205 rainbow_fg_term()

```
char* rainbow_fg_term (
            const char * s,
            double freq,
            size_t offset,
            size_t spread )
```

This is exactly like rainbow_fg(), except it uses colors that are closer to the standard 256-color values.

This prepends a color code to every character in the string. Multi-byte characters are handled properly by checking colr_mb_len(), and copying the bytes to the resulting string without codes between the multibyte characters.

The `CODE_RESET_ALL` code is appended to the result.

Parameters

| | | |
|---|---|---|
| `in` | *s* | The string to colorize.<br>Input *must be `null`-terminated*. |
| `in` | *freq* | Frequency ("tightness") for the colors. |
| `in` | *offset* | Starting offset in the rainbow. |
| `in` | *spread* | Number of characters per color. |

Returns

The allocated/formatted string on success.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

#### 0.6.2.6.206 rainbow_step()

```
RGB rainbow_step (
            double freq,
            size_t offset )
```

A single step in rainbow-izing produces the next color in the "rainbow" as an RGB value.

Parameters

| | | |
|---|---|---|
| `in` | *freq* | Frequency ("tightness") of the colors. |
| `in` | *offset* | Starting offset in the rainbow. |

Returns

An RGB value with the next "step" in the "rainbow".

Referenced by _rainbow().

#### 0.6.2.6.207 RGB_average()

```
unsigned char RGB_average (
            RGB rgb )
```

Return the average for an RGB value.

This is also it's "grayscale" value.

Parameters

| | | |
|---|---|---|
| in | *rgb* | The RGB value to get the average for. |

Returns

A value between `0`–`255`.

See also

RGB

Referenced by RGB_grayscale().

#### 0.6.2.6.208 RGB_eq()

```
bool RGB_eq (
            RGB a,
            RGB b )
```

Compare two RGB structs.

Parameters

| | | |
|---|---|---|
| in | *a* | First RGB value to check. |
| in | *b* | Second RGB value to check. |

Returns

`true` if a and b have the same `r`, g, and b values, otherwise `false`.

See also

RGB

Referenced by ColorValue_eq(), and ExtendedValue_from_RGB().

#### 0.6.2.6.209 RGB_from_BasicValue()

```
RGB RGB_from_BasicValue (
            BasicValue bval )
```

Return an RGB value from a known BasicValue.

Terminals use different values to render basic 3/4-bit escape-codes. The values returned from this function match the names found in `colr_name_data[]`.

Parameters

| | | |
|---|---|---|
| in | *bval* | A BasicValue to get the RGB value for. |

Returns

An RGB value that matches the BasicValue's color.

See also

RGB

#### 0.6.2.6.210 RGB_from_esc()

```
int RGB_from_esc (
            const char * s,
            RGB * rgb )
```

Convert an escape-code string (`char*`) to an actual RGB value.

Parameters

| | | |
|---|---|---|
| in | *s* | Escape-code string.<br>*Must be `null`-terminated.* |
| out | *rgb* | Pointer to an RGB struct to fill in the values for. |

Return values

| | |
|---|---|
| <*tt*>*0*</*tt*> | on success, with `rgb` filled with values. |
| *COLOR_INVALID* | on error (or if `s` is `NULL`). |
| *COLOR_INVALID_RANGE* | if any code numbers were outside of the range `0-255`. |

See also

RGB

#### 0.6.2.6.211 RGB_from_ExtendedValue()

```
RGB RGB_from_ExtendedValue (
            ExtendedValue eval )
```

Return an RGB value from a known ExtendedValue.

This is just a type/bounds-checked alias for `ext2rgb_map[eval]`.

Parameters

| | | |
|---|---|---|
| `in` | *eval* | An ExtendedValue to get the RGB value for. |

Returns

An RGB value from `ext2rgb_map[]`.

See also

RGB

#### 0.6.2.6.212 RGB_from_hex()

```
int RGB_from_hex (
            const char * hexstr,
            RGB * rgb )
```

Convert a hex color into an RGB value.

The format for hex strings can be one of:

- "[#]ffffff" (Leading hash symbol is optional)
- "[#]fff" (short-form)

Parameters

| | | |
|---|---|---|
| `in` | *hexstr* | String to check for hex values.<br>Input *must be `null`-terminated*. |
| `out` | *rgb* | Pointer to an RGB struct to fill in the values for. |

Return values

| | |
|---|---|
| *0* | on success, with `rgb` filled with the values. |
| *COLOR_INVALID* | for non-hex strings. |

See also

RGB

Referenced by ExtendedValue_from_hex(), RGB_from_hex_default(), and RGB_from_str().

#### 0.6.2.6.213 RGB_from_hex_default()

```
RGB RGB_from_hex_default (
            const char * hexstr,
            RGB default_value )
```

Convert a hex color into an RGB value, but use a default value when errors occur.

The format for hex strings can be one of:

- "[#]ffffff" (Leading hash symbol is optional)
- "[#]fff" (short-form)

Parameters

| | | |
|---|---|---|
| in | *hexstr* | String to check for RGB values.<br>Input *must be* `null`*-terminated*. |
| out | *default_value* | An RGB value to use when errors occur. |

Returns

A valid RGB value on success, or `default_value` on error.

See also

RGB
hex

#### 0.6.2.6.214 RGB_from_str()

```
int RGB_from_str (
            const char * arg,
            RGB * rgb )
```

Convert an RGB string (`char*`) into an RGB value.

The format for RGB strings can be one of:

- "RED,GREEN,BLUE"

- "RED GREEN BLUE"
- "RED:GREEN:BLUE"
- "RED;GREEN;BLUE" Or hex strings can be used:
- "#ffffff" (Leading hash symbol is **NOT** optional)
- "#fff" (short-form)

Parameters

| | | |
|---|---|---|
| `in` | *arg* | String to check for RGB values.<br>Input *must be* `null`*-terminated*. |
| `out` | *rgb* | Pointer to an RGB struct to fill in the values for. |

Return values

| | |
|---|---|
| *0* | on success, with `rgb` filled with the values. |
| *COLOR_INVALID* | for non-rgb strings. |
| *COLOR_INVALID_RANGE* | for rgb values outside of 0-255. |

See also

RGB

0.6.2.6.215 RGB_grayscale()

```
RGB RGB_grayscale (
            RGB rgb )
```

Return a grayscale version of an RGB value.

Parameters

| | | |
|---|---|---|
| `in` | *rgb* | The RGB value to convert. |

Returns

A grayscale RGB value.

See also

RGB

#### 0.6.2.6.216 RGB_inverted()

```
RGB RGB_inverted (
            RGB rgb )
```

Make a copy of an RGB value, with the colors "inverted" (like highlighting text in the terminal).

Parameters

| | | |
|---|---|---|
| in | *rgb* | The RGB value to invert. |

Returns

An "inverted" RGB value.

See also

RGB

#### 0.6.2.6.217 RGB_monochrome()

```
RGB RGB_monochrome (
            RGB rgb )
```

Convert an RGB value into either black or white, depending on it's average grayscale value.

Parameters

| | | |
|---|---|---|
| in | *rgb* | The RGB value to convert. |

Returns

Either `rgb(1, 1, 1)` or `rgb(255, 255, 255)`.

See also

RGB

#### 0.6.2.6.218 RGB_repr()

```
char* RGB_repr (
            RGB rgb )
```

Creates a string (`char*`) representation for an RGB value.

Allocates memory for the string representation.

Parameters

| | | |
|---|---|---|
| in | *rgb* | RGB struct to get the representation for. |

Returns

Allocated string for the representation.
*You must `free()` the memory allocated by this function.*

See also

RGB

#### 0.6.2.6.219 RGB_to_hex()

```
char* RGB_to_hex (
            RGB rgb )
```

Converts an RGB value into a hex string (`char*`).

Parameters

| | | |
|---|---|---|
| in | *rgb* | RGB value to convert. |

Returns

An allocated string.

*You must `free()` the memory allocated by this function.*

*If allocation fails, `NULL` is returned*.

See also

RGB

#### 0.6.2.6.220 RGB_to_str()

```
char* RGB_to_str (
            RGB rgb )
```

Convert an RGB value into a human-friendly RGB string (`char*`) suitable for input to RGB_from_↩ str().

Parameters

| | | |
|---|---|---|
| in | *rgb* | RGB value to convert. |

Returns

An allocated string in the form `"red;green;blue"`.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned.*

See also

RGB

#### 0.6.2.6.221 RGB_to_term_RGB()

```
RGB RGB_to_term_RGB (
            RGB rgb )
```

Convert an RGB value into it's nearest terminal-friendly RGB value.

This is a helper for the 'to_term' functions.

Parameters

| | | |
|---|---|---|
| in | *rgb* | RGB to convert. |

Returns

A new RGB with values close to a terminal code color.

See also

RGB

Referenced by ExtendedValue_from_RGB().

#### 0.6.2.6.222 StyleValue_eq()

```
bool StyleValue_eq (
            StyleValue a,
            StyleValue b )
```

Compares two StyleValues.

This is used to implement colr_eq().

Parameters

| | | |
|---|---|---|
| in | *a* | The first StyleValue to compare. |
| in | *b* | The second StyleValue to compare. |

Returns

`true` if they are equal, otherwise `false`.

See also

StyleValue

#### 0.6.2.6.223 StyleValue_from_esc()

```
StyleValue StyleValue_from_esc (
            const char * s )
```

Convert an escape-code string (`char*`) to an actual StyleValue enum value.

Parameters

| | | |
|---|---|---|
| in | *s* | Escape-code string.<br>*Must be `null`-terminated.* |

Return values

| | |
|---|---|
| *StyleValue* | value on success. |
| *STYLE_INVALID* | on error (or if `s` is `NULL`). |
| *STYLE_INVALID_RANGE* | if the code number was outside of the range `0-255`. |

See also

StyleValue

#### 0.6.2.6.224 StyleValue_from_str()

```
StyleValue StyleValue_from_str (
            const char * arg )
```

Convert a named argument to actual StyleValue enum value.

Parameters

| | | |
|---|---|---|
| in | *arg* | Style name to convert into a StyleValue. |

Returns

A usable StyleValue value on success, or STYLE_INVALID on error.

See also

StyleValue

#### 0.6.2.6.225 StyleValue_is_invalid()

```
bool StyleValue_is_invalid (
            StyleValue sval )
```

Determines whether a StyleValue is invalid.

Parameters

| | | |
|---|---|---|
| in | *sval* | A StyleValue to check. |

Returns

`true` if the value is considered invalid, otherwise `false`.

See also

StyleValue

#### 0.6.2.6.226 StyleValue_is_valid()

```
bool StyleValue_is_valid (
            StyleValue sval )
```

Determines whether a StyleValue is valid.

Parameters

| | | |
|---|---|---|
| in | *sval* | A StyleValue to check. |

Returns

`true` if the value is considered valid, otherwise `false`.

See also

StyleValue

#### 0.6.2.6.227 StyleValue_repr()

```
char* StyleValue_repr (
            StyleValue sval )
```

Creates a string (`char*`) representation of a StyleValue.

Parameters

| | | |
|---|---|---|
| `in` | *sval* | A StyleValue to get the value from. |

Returns

A pointer to an allocated string.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

StyleValue

#### 0.6.2.6.228 StyleValue_to_str()

```
char* StyleValue_to_str (
            StyleValue sval )
```

Create a human-friendly string (`char*`) representation for a StyleValue.

Parameters

| | | |
|---|---|---|
| `in` | *sval* | StyleValue to get the name for. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

StyleValue

#### 0.6.2.6.229 TermSize_repr()

```
char* TermSize_repr (
            TermSize ts )
```

Create a string (`char*`) representation for a TermSize.

Parameters

| | | |
|---|---|---|
| `in` | *ts* | TermSize to get the representation for. |

Returns

An allocated string with the result.
*You must `free()` the memory allocated by this function.*
*If allocation fails, `NULL` is returned*.

See also

TermSize

### 0.6.2.7 Variable Documentation

#### 0.6.2.7.1 colr_printf_esc_mod

```
int colr_printf_esc_mod
```

Integer to test for the presence of the "escaped output modifier" in colr_printf_handler.

This is set in colr_printf_register.

Referenced by colr_printf_handler(), and colr_printf_register().

## 0.7 Example Documentation

### 0.7.1 back_example.c

```c
#include "colr.h"

int main(void) {
    // Basic colors:
    char* s = colr_cat(
        fore(BLACK),
        back(RED), "This is a test",
        back(BLUE), " and only a test."
    );
    if (!s) return 1;
    printf("%s\n", s);
    free(s);

    // Color names:
    char* n = colr_cat(
        back("blue"),
        fore("white"),
        "This is blue."
    );
    if (!n) return 1;
    printf("%s\nThis is not.\n", n);
    free(n);

    // Extended (256) colors:
    char* e = colr_cat(fore(ext(0)), back(ext(35)), "Extended colors.\n");
    if (!e) return 1;
    printf("%s", e);
    free(e);

    // RGB (True Color) colors:
    char* r = colr_cat(back(rgb(35, 0, 155)), "RGB");
    if (!r) return 1;
    printf("%s\n", r);
    free(r);

    // Hex (RGB style) colors:
    char* h = colr_cat(
        back("#ff0000"), "Hex RGB\n",
        back(hex("fff")), fore(hex("000000")), "Hex macro RGB\n",
        back(hex_or("NOTHEX", rgb(255, 255, 255))), "Using default for bad hex str"
    );
    if (!h) return 1;
    printf("%s\n", h);
    free(h);

    // Hex (Closest ExtendedValue) colors:
    char* he = colr_cat(
        back(ext_hex("ff0000")), "Closest ExtendedValue Hex\n",
        back(ext_hex_or("NOTAHEX", ext(255))), "Using default for bad hex str"
    );
    if (!he) return 1;
    printf("%s\n", he);
    free(he);

    /*
        Colr() accepts a back() as one of it's arguments.
        The order does not matter.
```

```
    */
    char* colorized = colr_cat(
        Colr("This is red.\n", back(RED)),
        Colr("This is also red.\n", fore("white"), back("red")),
        "This is not."
    );
    if (!colorized) return 1;
    printf("%s\n", colorized);
    free(colorized);
}
```

### 0.7.2 ColorResult_example.c

```
#include "colr.h"

int main(void) {
    /*
        ColorResults mark an *allocated* string as "safe to free()" in the
        Colr macros/functions. You can wrap your own allocated strings by
        calling `ColrResult(mystring)`. Colr uses this behind the scenes to
        implement the Colr_join macro, which allows nested joins.
    */

    // Colr tries to make things easy, so you don't have to do this.
    // But if you *have to*, ColrResult will help you.
    // This example wouldn't need ColrResult if you used Colr_join instead,
    // which returns an allocated ColorResult itself.
    char* joined = colr_cat(
        ColrResult(colr_join(
            ColrResult(colr_join(
                ": ",
                Colr("debug", fore(GREEN)),
                Colr("This is a test.", fore(CYAN))
            )),
            "[",
            "]"
        )),
        "\nStack-allocated.",
        ColrResult(strdup("\nHeap-allocated for no reason."))
    );
    if (!joined) return EXIT_FAILURE;
    printf("%s\n", joined);
    // All your left with is the final allocated string result.
    free(joined);

    /*
        Without ColorResult/ColrResult, Colr will never call `free()` on your
        strings, or the strings created by Colr:
    */
    char* mine = strdup("I need this for later, don't free it.");
    if (!mine) return EXIT_FAILURE;
    char* colorized = colr(mine, fore(BLUE), back(WHITE));
    if (!colorized) return EXIT_FAILURE;
    printf("%s\n", colorized);
    // Your string is still good:
    printf("%s\n", mine);

    char* appended = colr_cat(colorized, "...still here.");
    if (!appended) return EXIT_FAILURE;
    printf("%s\n", appended);
    // The Colr-allocated string is still good:
```

```
    printf("%s\n", colorized);

    // Most colorization is a one-shot thing that doesn't need to stick
    // around, so these examples are here *just in case* you have to do this.
    // Watch these disappear when wrapped in a ColorResult and sent through
    // the colr functions/macros:
    char* final = colr_join(
        "\n",
        ColrResult(mine),
        ColrResult(colorized),
        ColrResult(appended)
    );
    if (!final) return EXIT_FAILURE;
    printf("%s\n", final);

    // All those allocations, and it's down to just the last call to colr_join().
    free(final);

    /*
        Colr_join() returns an allocated ColorResult itself, so if you were
        to use it outside of the colr macros/functions you would need to
        deal with printing/freeing it:
    */
    ColorResult* result = Colr_join(
        "\n",
        Colr("This is a line.", fore(ext_rgb(255, 128, 128))),
        ColrResult(colr_cat(
            Colr("This is another", style(UNDERLINE)),
            "."
        )),
        Colr_join("This is the final line.", "[", "]")
    );
    if (!result) return EXIT_FAILURE;
    // This actually compiles as: ColorResult_to_str(*result).
    printf("%s\n", colr_to_str(*result));

    // And, finally release the resources.
    // This actually ends up calling ColorResult_free(result) in the end:
    colr_free(result);

    /*
        Run this example through valgrind/libasan (-fsanitize=leak).
    */
}
```

### 0.7.3 colr_cat_example.c

```
#include "colr.h"

int main(void) {
    /*
        You can build your strings with colr_cat().
        Using a Colr (ColorText), or sprinkling fore(), back(), and style() calls,
        you can build multi-color strings and only worry about allocating/freeing
        the text.

        The order/number of arguments does not matter.
        colr_cat() accepts ColorTexts, ColorArgs, and strings (char*).
    */
    char *colorized = colr_cat(
        "This is plain.\n",
```

```
        Colr("This is styled.\n", fore(rgb(255, 0, 155))),
        fore(RED),
        "This was styled by the previous ColorArg.\n",
        NC,
        "This is normal because of the 'reset code' that came before it.\n",
        // See the colr_join example for more about this:
        Colr_join(Colr("This was joined", fore(RED)), "[", "]")
    );

    // Prints a colorized, joined, version of all the strings above.
    printf("%s\n", colorized);

    // Free the allocated result, no leaks.
    free(colorized);


    // Like I said before, if your text was allocated, you must free it.
    char *allocated;
    asprintf(&allocated, "\nThis is my string #%d\n", 1);

    char *colored = colr_cat(
        Colr(allocated, fore(ext(255)), style(UNDERLINE)),
        "This one should not be free'd though.\n"
    );
    printf("%s", colored);
    free(colored);
    free(allocated);

    /*
        For throw-away/nested results that will be used in ColrC functions/macros,
        you can use the Colr_cat variant.
    */
    colr_puts(Colr_cat("No leaks: ", Colr("see", fore(RED)), "?"));
}
```

### 0.7.4 Colr_example.c

```
#include "colr.h"

int main(void) {
    /*
        Colr() is for styling one piece of text.
        When combined with the colr_cat() macro it allows you to seperate colors/styles.
    */
    char* colorized = colr_cat(
        Colr("America ", fore(RED)),
        Colr("the ", fore(WHITE)),
        Colr("beautiful", fore(BLUE)),
        ".\n"
    );

    /*
        All of the Colr, fore, back, and style resources were free'd by 'colr'.
        You are responsible for the text and the resulting colorized string.
    */
    if (!colorized) return 1;
    printf("%s", colorized);
    free(colorized);

    /*
        There are three justification macros that make it easy to create
```

```
        ColorText's with center, left, or right-justified text.
    */
    char* just = colr_cat(
        Colr_center("This is centered.", 80, fore("lightblue")),
        "\n",
        Colr_ljust("This is on the left.", 38, fore(ext_hex("ff2525"))),
        "-----",
        Colr_rjust("This is on the right.", 38, fore(ext_rgb(255, 53, 125)))
    );
    if (!colorized) return 1;
    printf("%s\n", just);
    free(just);

    /*
        If you don't need several Colr() calls, there is a shortcut for safely
        creating colorized text using colr().
    */
    char* fast = colr(
        "Hello from ColrC.",
        fore("#2500FF"),
        back(ext_hex("#353535")),
        style(UNDERLINE)
    );
    if (!fast) return 1;
    printf("%s\n", fast);
    free(fast);
}
```

### 0.7.5 colr_join_example.c

```
#include "colr.h"

int main(void) {
    /*
        You can join things by a plain string or a colorized string.

        For the pieces, the order/number of arguments does not matter.
        colr_join() accepts ColorArgs, ColorResults, ColorTexts, and strings (char*).
    */
    char* colorized = colr_join(
        "\n",
        "This is a plain line.",
        Colr("This one is some kind of purple.", fore(rgb(125, 0, 155))),
        Colr("This one is bright.", style(BRIGHT)),
        "Another plain one, why not?"
    );
    if (!colorized) return 1;
    // Prints each colorized piece of text on it's own line:
    printf("%s\n", colorized);
    free(colorized);

    /*
        The joiner can be a ColorText, string, or ColorArg (though ColorArgs
        would be kinda useless).
    */
    char* final = colr_join(
        Colr(" <--> ", fore(ext_hex("#353535")), style(UNDERLINE)),
        "This",
        Colr(" that ", fore(RED)),
        "the other."
    );
```

```
    if (!final) return 1;
    // Prints each piece, joined by a colorized " <--> ".
    printf("%s\n", final);
    free(final);

    /*
        Nested joins can be achieved without leaking memory by using Colr_join().
        It wraps it's results in a ColorResult, which the colr macros are safe
        to `free()`.
    */
    colr_puts(
        Colr_join(
            " ",
            Colr_join(
                Colr("warning", fore(YELLOW)),
                "[",
                "]"
            ),
            Colr("This combination of calls should never leak.", fore(RED))
        )
    );
    /*
        Arrays of ColorText, ColorArgs, ColorResults, or strings can be used with
        colr_join_array().
    */
    char* joiner = " [and] ";
    ColorText* words[] = {
        Colr("this", fore(RED)),
        Colr("that", fore(hex("ff3599"))),
        Colr("the other", fore(BLUE), style(UNDERLINE)),
        // The last member must be NULL.
        NULL
    };
    char* s = colr_join_array(joiner, words);
    if (!s) {
        // Couldn't allocate memory for the final string.
        for (size_t i = 0; words[i]; i++) colr_free(words[i]);
        return 1;
    }
    printf("%s\n", s);
    free(s);

    // Don't forget to free your ColorResults/ColorTexts/ColorArgs.
    for (size_t i = 0; words[i]; i++) colr_free(words[i]);
}
```

### 0.7.6 colr_printf_example.c

```
#include "colr.h"

int main(void) {
    /*
        colr_printf registers a new format specifier, COLR_FMT_CHAR, to be used
        with printf. colr_printf acts like printf when called, except Colr
        object pointers can be passed directly, and their resources will be
        free()'d automatically.

        Notice that the Colr* macros/functions are used inside of the call,
        instead of the colr* (lowercase) macros/functions. This is because
        the Colr* versions all return an allocated ColorResult that will be
        automatically free()'d. Using the lowercase versions directly will cause
```

```
    a memory leak.
*/
colr_printf(
    "This is a Colr: %R\n",
    Colr("This", fore(RED))
);

/*
    Left/right justify work as expected, and a space can be used for
    center-justified text.
    %-NR : Left-justify to a width of N.
    %NR  : Right-justify to a width of N.
    % NR : Center-justify to a width of N.
*/
colr_printf(
    "%-10R | % 10R | %10R\n",
    Colr("Left", fore(RED)),
    Colr("Center", style(UNDERLINE)),
    Colr("Right", fore(BLUE))
);

/*
    The alternate-form for a Colr object is a string with no escape codes.

    %#R : Print the Colr object, but do not add escape codes.
*/
colr_printf(
    "   With colors: %R\nWithout colors: %#R\n",
    Colr("hello", fore(RED)),
    Colr("hello", fore(RED))
);

/*
    A custom modifier was added (‘/‘), to allow for escaped output.

    %/R : Print the Colr object, with the output string escaped.
*/
colr_printf(
    "        Normal: %R\n       Escaped: %/R\n",
    Colr("okay", fore(RED)),
    Colr("okay", fore(RED))
);

/*
    Other printf-like functions are available, like ‘sprintf‘, ‘snprintf‘,
    and ‘asprintf‘.
*/

// Better have room for the codes:
size_t possible_len = 10 + CODE_ANY_LEN;
char mystring[possible_len];
colr_sprintf(mystring, "%R", Colr("Again.", fore(RED),
  style(BRIGHT)));
puts(mystring);

// Ensure only a certain number of bytes are written:
colr_snprintf(mystring, possible_len, "%R", Colr("Safe?",
  fore(BLUE)));
puts(mystring);

// Allocate the string, and then fill it:
char* myalloced = NULL;
if (colr_asprintf(&myalloced, "This: %R", Colr("Hah!", fore("dimgrey"))) < 1) {
```

```
        // Failed to allocate.
        return EXIT_FAILURE;
    }
    puts(myalloced);
    free(myalloced);
}
```

### 0.7.7 colr_replace_all_example.c

```c
#include "colr.h"

int main(void) {

    // The string we are modifying.
    char* mystring = "This was foo. I mean foo.";
    char* pattern = "foo";

    /*
        Replace a string with a string.
    */
    char* replaced = colr_replace_all(
        mystring,
        pattern,
        "replacement"
    );
    // Failed to allocate for new replaced string?
    if (!replaced) return EXIT_FAILURE;
    // Print the result and `free()` it.
    puts(replaced);
    free(replaced);

    /*
        Replace a string with a ColorText.
    */
    replaced = colr_replace_all(
        mystring,
        pattern,
        Colr("replacement", fore(RED))
    );
    // Failed to allocate for new replaced string?
    if (!replaced) return EXIT_FAILURE;
    // Print the result and `free()` it.
    puts(replaced);
    free(replaced);

    /*
        Replace a string with a ColorResult.
    */
    replaced = colr_replace_all(
        mystring,
        pattern,
        Colr_join(
            " ",
            Colr("really", style(BRIGHT)),
            Colr("replaced", fore(BLUE))
        )
    );
    // Failed to allocate for new replaced string?
    if (!replaced) return EXIT_FAILURE;
    // Print the result and `free()` it.
    puts(replaced);
```

```
free(replaced);

/*
    Replace a string with a ColorResult.
*/
char* mytemplate = "This REDis " NC "kinda REDuseful" NC "?";
replaced = colr_replace_all(
    mytemplate,
    "RED",
    fore(RED)
);
// Failed to allocate for new replaced string?
if (!replaced) return EXIT_FAILURE;
// Print the result and `free()` it.
puts(replaced);
free(replaced);

/*
    Replace a `NULL`-terminated array of regex matches with a ColorText.
*/
char* mymatchstring = "I think this is a beautiful thing.";
regex_t pat;
if (regcomp(&pat, "th[a-z]+", REG_EXTENDED)) {
    regfree(&pat);
    fprintf(stderr, "Failed to compile regex!\n");
    return EXIT_FAILURE;
}
// `colr_re_matches` returns a `NULL`-terminated array of regex matches.
regmatch_t** matches = colr_re_matches(mymatchstring, &pat);
// We don't need the pattern anymore, `free()` it.
regfree(&pat);
if (!matches) {
    // Impossible (for this example).
    colr_free(matches);
    fprintf(stderr, "Failed to match anything!\n");
    return EXIT_FAILURE;
}
replaced = colr_replace_all(mymatchstring, matches, Colr("uhhh",
  fore(RED)));
// We don't need the matches anymore, `free()` them.
// You must use colr_free_re_matches() or the colr_free() macro.
colr_free(matches);
if (!replaced) return EXIT_FAILURE;
// Print the result and `free()` it.
puts(replaced);
free(replaced);

/*
    Replace a compiled regex pattern with a ColorText.
*/
char* mypatstring = "I think this is a beautiful thing.";
regex_t mypat;
if (regcomp(&mypat, "th[a-z]+", REG_EXTENDED)) {
    regfree(&mypat);
    fprintf(stderr, "Failed to compile regex!\n");
    return EXIT_FAILURE;
}
replaced = colr_replace_all(mypatstring, &mypat, Colr("..uh",
  fore(BLUE)));
// We don't need the pattern anymore, `free()` it.
regfree(&mypat);
if (!replaced) return EXIT_FAILURE;
// Print the result and `free()` it.
```

```
    puts(replaced);
    free(replaced);
    return EXIT_SUCCESS;
}
```

### 0.7.8 colr_replace_example.c

```
#include "colr.h"

int main(void) {

    // The string we are modifying.
    char* mystring = "This is a foo line.";
    char* pattern = "foo";

    /*
        Replace a string with a string.
    */
    char* replaced = colr_replace(
        mystring,
        pattern,
        "replaced"
    );
    // Failed to allocate for new replaced string?
    if (!replaced) return EXIT_FAILURE;
    // Print the result and `free()` it.
    printf("%s\n", replaced);
    free(replaced);

    /*
        Replace a string with a ColorText.
    */
    replaced = colr_replace(
        mystring,
        pattern,
        Colr("replaced", fore(RED))
    );
    // Failed to allocate for new replaced string?
    if (!replaced) return EXIT_FAILURE;
    // Print the result and `free()` it.
    printf("%s\n", replaced);
    free(replaced);

    /*
        Replace a string with a ColorResult.
    */
    replaced = colr_replace(
        mystring,
        pattern,
        Colr_join(
            " ",
            Colr("really", style(BRIGHT)),
            Colr("replaced", fore(BLUE))
        )
    );
    // Failed to allocate for new replaced string?
    if (!replaced) return EXIT_FAILURE;
    // Print the result and `free()` it.
    printf("%s\n", replaced);
    free(replaced);
```

```
    /*
        Replace a string with a ColorResult.
    */
    char* mytemplate = "This is REDuseful?" NC;
    replaced = colr_replace(
        mytemplate,
        "RED",
        fore(RED)
    );
    // Failed to allocate for new replaced string?
    if (!replaced) return EXIT_FAILURE;
    // Print the result and ‘free()‘ it.
    printf("%s\n", replaced);
    free(replaced);

    /*
        Replace a compiled regex pattern with a ColorText.
    */
    char* mypatstring = "I think this is a beautiful thing.";
    regex_t mypat;
    if (regcomp(&mypat, "th[a-z]+", REG_EXTENDED)) {
        regfree(&mypat);
        fprintf(stderr, "Failed to compile regex!\n");
        return EXIT_FAILURE;
    }
    replaced = colr_replace(mypatstring, &mypat, Colr("know",
      fore(BLUE)));
    // We don’t need the pattern anymore, ‘free()‘ it.
    regfree(&mypat);
    if (!replaced) return EXIT_FAILURE;
    // Print the result and ‘free()‘ it.
    puts(replaced);
    free(replaced);

    return EXIT_SUCCESS;
}
```

### 0.7.9 colr_replace_re_all_example.c

```
#include "colr.h"

int main(void) {
    /*
        If you already have a ‘NULL‘-terminated array of ‘regmatch_t‘ (‘regmatch_t**‘),
        a single ‘regex_t‘, or a compiled regex pattern (‘regex_t‘),
        you can use colr_replace() or colr_replace_all().
        This macro (colr_replace_re_all) is for string patterns.
    */

    // The string we are modifying.
    char* mystring = "This was foo, and I mean foo.";
    char* pattern = "fo{2}";

    /*
        Replace all regex matches with a string.
    */
    char* replaced = colr_replace_re_all(
        mystring,
        pattern,
        "replaced",
        0
```

```
    );
    // Failed to allocate for new replaced string?
    if (!replaced) return EXIT_FAILURE;
    // Print the result and `free()` it.
    printf("%s\n", replaced);
    free(replaced);

    /*
        Replace all regex matches with a ColorText.
    */
    replaced = colr_replace_re_all(
        mystring,
        pattern,
        Colr("replaced", fore(RED)),
        REG_ICASE
    );
    // Failed to allocate for new replaced string?
    if (!replaced) return EXIT_FAILURE;
    // Print the result and `free()` it.
    printf("%s\n", replaced);
    free(replaced);

    /*
        Replace all regex matches with a ColorResult.
    */
    replaced = colr_replace_re_all(
        mystring,
        pattern,
        Colr_join(
            " ",
            Colr("really", style(BRIGHT)),
            Colr("replaced", fore(BLUE))
        ),
        0
    );
    // Failed to allocate for new replaced string?
    if (!replaced) return EXIT_FAILURE;
    // Print the result and `free()` it.
    printf("%s\n", replaced);
    free(replaced);

    /*
        Replace all regex matches with a ColorResult.
    */
    char* mytemplate = "This REDis " NC "kinda REDuseful?" NC;
    replaced = colr_replace_re_all(
        mytemplate,
        "RED",
        fore(RED),
        0
    );
    // Failed to allocate for new replaced string?
    if (!replaced) return EXIT_FAILURE;
    // Print the result and `free()` it.
    printf("%s\n", replaced);
    free(replaced);

    return EXIT_SUCCESS;
}
```

### 0.7.10 colr_replace_re_example.c

```c
#include "colr.h"

int main(void) {
    /*
        If you already have a `NULL`-terminated array of `regmatch_t` (`regmatch_t**`),
        a single `regex_t`, or a compiled regex pattern (`regex_t`),
        you can use colr_replace() or colr_replace_all().
        This macro (colr_replace_re_all) is for string patterns.
    */


    // The string we are modifying.
    char* mystring = "This is a foo line.";
    char* pattern = "fo{2}";

    /*
        Replace a regex match with a string.
    */
    char* replaced = colr_replace_re(
        mystring,
        pattern,
        "replaced",
        0
    );
    // Failed to allocate for new replaced string?
    if (!replaced) return EXIT_FAILURE;
    // Print the result and `free()` it.
    printf("%s\n", replaced);
    free(replaced);

    /*
        Replace a regex match with a ColorText.
    */
    replaced = colr_replace_re(
        mystring,
        pattern,
        Colr("replaced", fore(RED)),
        REG_ICASE
    );
    // Failed to allocate for new replaced string?
    if (!replaced) return EXIT_FAILURE;
    // Print the result and `free()` it.
    printf("%s\n", replaced);
    free(replaced);

    /*
        Replace a regex match with a ColorResult.
    */
    replaced = colr_replace_re(
        mystring,
        pattern,
        Colr_join(
            " ",
            Colr("really", style(BRIGHT)),
            Colr("replaced", fore(BLUE))
        ),
        0
    );
    // Failed to allocate for new replaced string?
    if (!replaced) return EXIT_FAILURE;
    // Print the result and `free()` it.
```

```
    printf("%s\n", replaced);
    free(replaced);

    /*
        Replace a regex match with a ColorResult.
    */
    char* mytemplate = "This is REDuseful" NC "?";
    replaced = colr_replace_re(
        mytemplate,
        "RED",
        fore(RED),
        0
    );
    // Failed to allocate for new replaced string?
    if (!replaced) return EXIT_FAILURE;
    // Print the result and ‘free()‘ it.
    printf("%s\n", replaced);
    free(replaced);

    return EXIT_SUCCESS;
}
```

### 0.7.11 fore_example.c

```
#include "colr.h"

int main(void) {
    // Basic colors:
    char* s = colr_cat(
        fore(RED),
        "This is a test",
        fore(BLUE),
        " and only a test."
    );
    printf("%s\n", s);
    free(s);

    // Color names:
    char* n = colr_cat(
        fore("red"),
        "This is red."
    );
    printf("%s\n", n);
    free(n);

    // Extended (256) colors:
    char* e = colr_cat(fore(ext(35)), "Extended colors.");
    printf("%s\n", e);
    free(e);

    // RGB (True Color) colors:
    char* r = colr_cat(fore(rgb(35, 0, 155)), "RGB");
    printf("%s\n", r);
    free(r);

    /*
        Colr() accepts a fore() as one of it’s arguments.
        The order does not matter.
    */
    char* mystr = colr_cat(
        Colr("This is red.", fore(RED)),
```

```
        Colr("This is also red.", back("white"), fore("red")),
        "This is not.\n"
    );
    printf("%s\n", mystr);
    free(mystr);
}
```

### 0.7.12 simple_example.c

```
#include "colr.h"

int main(int argc, char** argv) {
    // Print-related macros, using Colr() to build colorized text:
    puts("\nColrC supports ");
    colr_puts(Colr_join(
        ", ",
        Colr("basic", fore(WHITE)),
        Colr("extended (256)", fore(ext(155))),
        Colr("rgb", fore(rgb(155, 25, 195))),
        Colr("hex", fore(hex("#ff00bb"))),
        Colr("extended hex", fore(ext_hex("#ff00bb"))),
        Colr("color names", fore("dodgerblue"), back("aliceblue")),
        Colr("and styles.", style(BRIGHT))
    ));

    colr_puts(
        "Strings and ",
        Colr("colors", fore(LIGHTBLUE)),
        " can be mixed in any order."
    );

    // Create a string, using colr(), instead of colr_puts() or colr_print().
    char* mystr = colr("Don't want to print this.", style(UNDERLINE));
    printf("\nNow I do: %s\n", mystr);
    free(mystr);

    // Concatenate existing strings with ColrC objects.
    // Remember that the colr macro free ColrC objects, not strings.
    // So I'm going to use the Colr* macros inside of this call (not colr*).
    char* catted = colr_cat(
        "Exhibit: ",
        Colr("b", fore(BLUE)),
        "\nThe ColorText/Colr was released."
    );
    puts(catted);
    free(catted);

    // Create a ColorText, on the heap, for use with colr_cat(), colr_print(),
    // or colr_puts().
    ColorText* ctext = NULL;
    if (argc == 1) {
        ctext = Colr("<nothing>", fore(RED));
    } else {
        ctext = Colr(argv[1], fore(GREEN));
    }
    char* userstr = colr_cat("Argument: ", ctext);
    puts(userstr);
    // colr_cat() already called ColorText_free(ctext).
    free(userstr);

    // Create a joined string (a "[warning]" label).
```

```
    char* warning_label = colr_join(Colr("warning", fore(YELLOW)), "[", "]");
    // Simulate multiple uses of the string.
    for (int i = 1; i < 4; i++) printf("%s This is #%d\n", warning_label, i);
    // Okay, now we're done with the colorized string.
    free(warning_label);

    // Colorize an existing string by replacing a word.
    char* logtext = "[warning] This is an awesome warning.";
    char* colorized = colr_replace(
        logtext,
        "warning",
        Colr("warning", fore(YELLOW))
    );
    // Failed to allocate for new string?
    if (!colorized) return EXIT_FAILURE;
    puts(colorized);
    // You have to free the resulting string.
    free(colorized);

    // Or colorize an existing string by replacing a regex pattern.
    colorized = colr_replace_re(
        logtext,
        "\\[\\w+\\]",
        Colr_join(
            Colr("ok", style(BRIGHT)),
            "(",
            ")"
        ),
        REG_EXTENDED
    );
    if (!colorized) return EXIT_FAILURE;
    puts(colorized);
    free(colorized);

    // Or maybe you want to replace ALL of the occurrences?
    char* logtext2 = "[warning] This is an awesome warning.";
    // There is also a colr_replace_re_all() if you'd rather use a regex pattern.
    char* colorizedall = colr_replace_all(
        logtext2,
        "warning",
        Colr("WARNING", fore(YELLOW))
    );
    // Failed to allocate for new string?
    if (!colorizedall) return EXIT_FAILURE;
    puts(colorizedall);
    // You have to free the resulting string.
    free(colorizedall);

}
```

### 0.7.13 style_example.c

```
#include "colr.h"

int main(void) {
    /*
        Styles can be given as a StyleValue, or a style name (see style_names).
    */
    char* s = colr_cat(
        style("bright"), "This is a test ",
        style(UNDERLINE), "and only a test."
```

```
    );
    printf("%s\n", s);
    free(s);

    /*
        Colr accepts a style() as one of it's arguments.
        The order does not matter.
    */
    char* mystr = colr_cat(
        Colr("THIS IS BOLD.\n", style(BRIGHT)),
        "This is not."
    );
    printf("%s\n", mystr);
    free(mystr);

}
```

# Index