



## 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	1
0.1.1	Getting Started	1
0.1.1.1	Including	1
0.1.1.2	Compiling	2
0.1.1.3	Files	2
0.1.1.4	Example Usage	2
0.1.1.5	Why	5
0.1.1.6	Future	5
0.2	Development	5
0.2.1	ColrC Development	5
0.2.2	Dependencies	6
0.2.2.1	System	6
0.2.2.2	Python	7
0.2.3	Tests	7
0.2.3.1	About	7
0.2.3.2	Basic Test:	7
0.2.3.3	Memcheck Test:	8
0.2.3.4	Quick Testing	8
0.2.3.5	Test Everything	8
0.2.3.6	Test Tool	8
0.2.4	Make	9
0.2.4.1	ColrC Make Targets	9
0.2.4.2	Build	9

0.2.4.3	Test	10
0.2.4.4	Document	10
0.2.4.5	Examples	10
0.2.5	Tools	11
0.2.5.1	ColrC Tools	11
0.2.6	Examples	12
0.2.6.1	ColrC Examples	12
0.2.7	Compatibility	13
0.2.7.1	About	13
0.2.7.2	Porting	13
0.2.7.3	Windows	13
0.3	Tool	13
0.3.1	About	13
0.3.1.1	Colorizing Text	14
0.3.1.2	Rainbows	14
0.3.1.3	Stripping Colorized Output	15
0.3.1.4	Inspecting Colorized Output	15
0.3.1.5	Translating Color Codes	15
0.3.2	Tool Building	16
0.3.2.1	Build	16
0.3.2.2	Install	16
0.3.2.3	Uninstall	16
0.4	File Index	17
0.4.1	File List	17
0.5	File Documentation	17
0.5.1	colr.c File Reference	17
0.5.1.1	Detailed Description	28
0.5.1.2	Function Documentation	28
0.5.1.3	Variable Documentation	150

0.5.2	colr.h File Reference	152
0.5.2.1	Detailed Description	170
0.5.2.2	Data Structure Documentation	170
0.5.2.3	Macro Definition Documentation	175
0.5.2.4	Typedef Documentation	228
0.5.2.5	Enumeration Type Documentation	228
0.5.2.6	Function Documentation	229
0.5.2.7	Variable Documentation	350
0.6	Example Documentation	351
0.6.1	back_example.c	351
0.6.2	ColorResult_example.c	352
0.6.3	colr_cat_example.c	353
0.6.4	Colr_example.c	354
0.6.5	colr_join_example.c	355
0.6.6	colr_printf_example.c	356
0.6.7	colr_replace_all_example.c	358
0.6.8	colr_replace_example.c	360
0.6.9	colr_replace_re_all_example.c	361
0.6.10	colr_replace_re_example.c	363
0.6.11	fore_example.c	364
0.6.12	simple_example.c	365
0.6.13	style_example.c	366
	Index	369



## 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, colored, 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("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
<a href="#">colr.h</a>	The interface to ColrC.
<a href="#">colr.c</a>	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 colored 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	<a href="#">Colr_example.c</a>
colr_cat	<a href="#">colr_cat_example.c</a>
colr_join	<a href="#">colr_join_example.c</a>
colr_replace	<a href="#">colr_replace_example.c</a>
colr_replace_re	<a href="#">colr_replace_re_↵ example.c</a>
fore	<a href="#">fore_example.c</a>
back	<a href="#">back_example.c</a>
style	<a href="#">style_example.c</a>

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 its "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 its 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 docshhtml`
  - 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 little 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 Tool

### 0.3.1 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:

### 0.3.1.1 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
```

### 0.3.1.2 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
```

### 0.3.1.3 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.

### 0.3.1.4 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
```

### 0.3.1.5 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
```

## 0.3.2 Tool Building

### 0.3.2.1 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.

### 0.3.2.2 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.

### 0.3.2.3 Uninstall

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

```
make uninstall
```

## 0.4 File Index

### 0.4.1 File List

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

<a href="#">colr.c</a>	Implements everything in the <a href="#">colr.h</a> header . . . . .	17
<a href="#">colr.h</a>	Declarations for ColrC functions, enums, structs, etc . . . . .	152

## 0.5 File Documentation

### 0.5.1 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.*
  - 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 \*joiner, void \*ps)
 

*Join an array of strings (`char*`), `ColorArgs`, or `ColorTexts` by another string (`char*`), `ColorArg`, or `ColorText`.*
  - char \* `colr_join_arrayn` (void \*joiner, 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_just` (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 *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](#).*

#### 0.5.1.1 Detailed Description

Implements everything in the [colr.h](#) header.

#### 0.5.1.2 Function Documentation

##### 0.5.1.2.1 [\\_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	<a href="#">p</a>	Pointer to a heap-allocated object.
----	-------------------	-------------------------------------

##### 0.5.1.2.2 [\\_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	<i>p</i>	The pointer to check.
----	----------	-----------------------

## Returns

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

0.5.1.2.3 `_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 `ColorArgs`, `ColorResults`, or `ColorTexts` 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	<i>joinerp</i>	The joiner (any <code>ColorArg</code> , <code>ColorResult</code> , <code>ColorText</code> , or string).
in	...	Zero or more <code>ColorArgs</code> , <code>ColorResults</code> , <code>ColorTexts</code> , 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`.*

0.5.1.2.4 `_colr_join_array_length()`

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

Determine the length of a `NULL`-terminated array of strings (`char*`), `ColorArgs`, `ColorResults`, or `ColorTexts`.

## Warning

This is for internal use only.

## Parameters

in	<i>ps</i>	A NULL-terminated array of <a href="#">ColorArgs</a> , <a href="#">ColorResults</a> , <a href="#">ColorTexts</a> , or strings (char*).
----	-----------	--

## Returns

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

Referenced by `colr_join_array()`.

0.5.1.2.5 `_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\*), [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	<i>joinerp</i>	The joiner (any <a href="#">ColorArg</a> , <a href="#">ColorResult</a> , <a href="#">ColorText</a> , or string).
in	<i>ps</i>	An array of pointers to <a href="#">ColorArgs</a> , <a href="#">ColorResults</a> , <a href="#">ColorTexts</a> , or strings. The array must have NULL as the last item if count is greater than the total number of items.
in	<i>count</i>	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()`.

#### 0.5.1.2.6 `_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	<i>joinerp</i>	The joiner (any <a href="#">ColorArg</a> , <a href="#">ColorText</a> , or string (char*)).
in	<i>args</i>	A <code>va_list</code> with zero or more <a href="#">ColorArgs</a> , <a href="#">ColorTexts</a> , 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_RESET_LEN`.

#### See also

`_colr`

Referenced by `_colr_join()`.

#### 0.5.1.2.7 `_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	<i>p</i>	A <a href="#">ColorArg</a> pointer, <a href="#">ColorResult</a> pointer, <a href="#">ColorText</a> 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()`.

0.5.1.2.8 `_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	<i>p</i>	A <a href="#">ColorArg</a> pointer, <a href="#">ColorResult</a> pointer, <a href="#">ColorText</a> 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](#)

0.5.1.2.9 `_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	<i>p</i>	A <a href="#">ColorArg</a> pointer, <a href="#">ColorResult</a> pointer, <a href="#">ColorText</a> 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.*

0.5.1.2.10 `_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	<i>fmter</i>	A formatter function ( <code>RGB_fmter</code> ) that can create escape codes from <a href="#">RGB</a> values.
in	<i>s</i>	The string to "rainbowize". Input <i>must be null-terminated</i> .
in	<i>freq</i>	The "tightness" for colors.
in	<i>offset</i>	The starting offset into the rainbow.
in	<i>spread</i>	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()`.

## 0.5.1.2.11 ArgType\_eq()

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

Compares two ArgTypes.

This is used to implement [colr\\_eq\(\)](#).

Parameters

in	<i>a</i>	The first ArgType to compare.
in	<i>b</i>	The second ArgType to compare.

Returns

true if they are equal, otherwise false.

## 0.5.1.2.12 ArgType\_repr()

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

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

Parameters

in	<i>type</i>	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\(\)](#).

## 0.5.1.2.13 ArgType\_to\_str()

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

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



## Parameters

in	<i>type</i>	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().

## 0.5.1.2.14 BasicValue\_eq()

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

Compares two BasicValues.

This is used to implement [colr\\_eq\(\)](#).

## Parameters

in	<i>a</i>	The first BasicValue to compare.
in	<i>b</i>	The second BasicValue to compare.

## Returns

true if they are equal, otherwise false.

## See also

[BasicValue](#)

## 0.5.1.2.15 BasicValue\_from\_esc()

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

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

## Parameters

in	<i>s</i>	Escape-code string. <i>Must be null-terminated.</i>
----	----------	--

## Return values

<i>BasicValue</i>	value on success.
<i>BASIC_INVALID</i>	on error (or if <i>s</i> is NULL).
<i>BASIC_INVALID_RANGE</i>	if the code number was outside of the range 0–255.

See also

[BasicValue](#)

## 0.5.1.2.16 BasicValue\_from\_str()

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

Convert named argument to an actual BasicValue enum value.

## Parameters

in	<i>arg</i>	Color name to find the BasicValue for.
----	------------	--

## Returns

BasicValue value on success, or BASIC\_INVALID on error.

See also

[BasicValue](#)

## 0.5.1.2.17 BasicValue\_is\_invalid()

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

Determines whether a BasicValue is invalid.

## Parameters

in	<i>bval</i>	A BasicValue to check.
----	-------------	------------------------

## Returns

true if the value is considered invalid, otherwise false.

## See also

[BasicValue](#)

Referenced by `ExtendedValue_from_BasicValue()`.

0.5.1.2.18 `BasicValue_is_valid()`

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

Determines whether a `BasicValue` is valid.

## Parameters

in	<i>bval</i>	A <code>BasicValue</code> to check.
----	-------------	-------------------------------------

## Returns

true if the value is considered valid, otherwise false.

## See also

[BasicValue](#)

0.5.1.2.19 `BasicValue_repr()`

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

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

## Parameters

in	<i>bval</i>	A <code>BasicValue</code> 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](#)

#### 0.5.1.2.20 BasicValue\_to\_ansi()

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

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

Parameters

in	<i>type</i>	ArgType (FORE/BACK).
in	<i>bval</i>	BasicValue to convert.

Returns

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

See also

[BasicValue](#)

Referenced by `format_bg()`, and `format_fg()`.

#### 0.5.1.2.21 BasicValue\_to\_str()

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

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

Parameters

in	<i>bval</i>	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](#)

#### 0.5.1.2.22 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](#)

#### 0.5.1.2.23 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	<i>a</i>	First <a href="#">ColorArg</a> to compare.
in	<i>b</i>	Second <a href="#">ColorArg</a> to compare.

Returns

true if they are equal, otherwise false.

See also

[ColorArg](#)

Referenced by [ColorText\\_has\\_arg\(\)](#).

## 0.5.1.2.24 ColorArg\_example()

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

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

## Parameters

in	<i>carg</i>	A <a href="#">ColorArg</a> to get an example string for.
in	<i>colored</i>	Whether to include a colored example. If set to <code>false</code> , 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](#)

## 0.5.1.2.25 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	<i>p</i>	<a href="#">ColorArg</a> 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()`.

## 0.5.1.2.26 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	<i>type</i>	ArgType (FORE, BACK, STYLE).
in	<i>value</i>	BasicValue to use.

Returns

A [ColorArg](#), with the `.value.type` member possibly set to TYPE\_INVALID.

See also

[ColorArg](#)

## 0.5.1.2.27 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	<i>s</i>	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()`.

#### 0.5.1.2.28 `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	<i>type</i>	ArgType (FORE, BACK, STYLE).
in	<i>value</i>	ExtendedValue to use.

Returns

A [ColorArg](#), with the `.value.type` member possibly set to `TYPE_INVALID`.

See also

[ColorArg](#)

#### 0.5.1.2.29 `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	<i>type</i>	ArgType (FORE, BACK, STYLE).
in	<i>value</i>	<a href="#">RGB</a> struct to use.



Returns

A [ColorArg](#), with the `.value.type` member possibly set to `TYPE_INVALID`.

See also

[ColorArg](#)

#### 0.5.1.2.30 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	<i>type</i>	ArgType (FORE, BACK, STYLE).
in	<i>colorname</i>	A known color name/style.

Returns

A [ColorArg](#) struct with usable values.

See also

[ColorArg](#)

#### 0.5.1.2.31 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	<i>type</i>	ArgType (FORE, BACK, STYLE).
in	<i>value</i>	StyleValue to use.

## Returns

A [ColorArg](#), with the `.value.type` member possibly set to `TYPE_INVALID`.

## See also

[ColorArg](#)

0.5.1.2.32 `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	<i>type</i>	ArgType value, to mark the type of <a href="#">ColorArg</a> .
in	<i>colrtype</i>	ColorType value, to mark the type of <a href="#">ColorValue</a> .
in	<i>p</i>	A pointer to either a BasicValue, ExtendedValue, or a <a href="#">RGB</a> .

## 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](#)

0.5.1.2.33 `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	<i>carg</i>	A <a href="#">ColorArg</a> 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()`.

0.5.1.2.34 `ColorArg_is_invalid()`

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

Checks to see if a [ColorArg](#) holds an invalid value.

## Parameters

in	<i>carg</i>	<a href="#">ColorArg</a> struct to check.
----	-------------	---

## Returns

true if the value is invalid, otherwise false.

## See also

[ColorArg](#)

0.5.1.2.35 `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	<i>p</i>	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()`.

0.5.1.2.36 `ColorArg_is_valid()`

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

Checks to see if a [ColorArg](#) holds a valid value.

## Parameters

in	<i>carg</i>	<a href="#">ColorArg</a> struct to check.
----	-------------	---

## Returns

true if the value is valid, otherwise false.

## See also

[ColorArg](#)

0.5.1.2.37 `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	<i>carg</i>	<a href="#">ColorArg</a> 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()`.

0.5.1.2.38 `ColorArg_repr()`

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

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

Allocates memory for the string representation.

## Parameters

in	<i>carg</i>	<a href="#">ColorArg</a> 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()`.

0.5.1.2.39 `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	<i>carg</i>	<a href="#">ColorArg</a> to get the ArgType and <a href="#">ColorValue</a> 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()`.

0.5.1.2.40 `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	<i>dest</i>	Destination for the escape code string. <i>Must have room for the code type being used.</i> See <a href="#">ColorArg_length()</a> for determining the size needed.
in	<i>carg</i>	<a href="#">ColorArg</a> to get the ArgType and <a href="#">ColorValue</a> from.

## Returns

true if the [ColorArg](#) was valid, otherwise false.

## See also

[ColorArg](#)

## 0.5.1.2.41 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	<i>carg</i>	<a href="#">ColorArg</a> 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()`.

## 0.5.1.2.42 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	<i>ps</i>	A pointer to an array of <code>ColorArgs</code> , where <code>NULL</code> is the last item.
----	-----------	---

## 0.5.1.2.43 ColorArgs\_array\_repr()

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

Creates a string representation for an array of [ColorArg](#) pointers.

## Parameters

in	<i>lst</i>	The <a href="#">ColorArg</a> array to create the representation for (ColorArg**).
----	------------	---

## Returns

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

## 0.5.1.2.44 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	<i>s</i>	A string to get the escape-codes from. <i>Must be null-terminated.</i>
in	<i>unique</i>	Whether to only include <i>unique</i> 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

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

## 0.5.1.2.45 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()`.

0.5.1.2.46 `ColorJustify_eq()`

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

Compares two [ColorJustify](#) structs.

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

## Parameters

in	<i>a</i>	First <a href="#">ColorJustify</a> to compare.
in	<i>b</i>	Second <a href="#">ColorJustify</a> to compare.

## Returns

true if they are equal, otherwise false.

## See also

[ColorJustify](#)

0.5.1.2.47 `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	<i>cjust</i>	The <a href="#">ColorJustify</a> to check.
----	--------------	--

## Returns

true if the [ColorJustify](#) is empty, otherwise false.

## See also

[ColorJustify](#)  
[ColorJustify\\_empty](#)

Referenced by [ColorText\\_is\\_empty\(\)](#), and [ColorText\\_length\(\)](#).

0.5.1.2.48 [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	<i>method</i>	<a href="#">ColorJustifyMethod</a> to use.
in	<i>width</i>	Width for justification. If 0 is given, <a href="#">ColorText</a> will use the width from <a href="#">colr_term_size()</a> .
in	<i>padchar</i>	Padding character to use. If 0 is given, the default, space (" "), is used.

## Returns

An initialized [ColorJustify](#).

0.5.1.2.49 [ColorJustify\\_repr\(\)](#)

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

Creates a string (char\*) representation for a [ColorJustify](#).

Allocates memory for the string representation.

## Parameters

in	<i>cjust</i>	<a href="#">ColorJustify</a> 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()`.

0.5.1.2.50 `ColorJustifyMethod_repr()`

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

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

Allocates memory for the string representation.

## Parameters

in	<i>meth</i>	<code>ColorJustifyMethod</code> 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()`.

0.5.1.2.51 `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()`.

0.5.1.2.52 `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	<i>a</i>	First <code>ColorResult</code> to compare.
in	<i>b</i>	Second <code>ColorResult</code> to compare.

Returns

true if they are equal, otherwise false.

See also

[ColorResult](#)

0.5.1.2.53 `ColorResult_free()`

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

Free allocated memory for a `ColorResult` and its `.result` member.

Parameters

in	<i>p</i>	A <code>ColorResult</code> with a NULL or heap-allocated <code>.result</code> 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()`.

0.5.1.2.54 `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	<i>p</i>	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()`.

0.5.1.2.55 `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	<i>cres</i>	A <a href="#">ColorResult</a> 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()`.

## 0.5.1.2.56 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](#)

## 0.5.1.2.57 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 <a href="#">ColorResult</a> 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()`.

## 0.5.1.2.58 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	<i>cres</i>	A <a href="#">ColorResult</a> 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](#)

## 0.5.1.2.59 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	<i>cres</i>	A <a href="#">ColorResult</a> to use.
----	-------------	---------------------------------------

## Returns

A stringified-version of 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()`.

#### 0.5.1.2.60 `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()`.

#### 0.5.1.2.61 `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	<i>p</i>	Pointer to <a href="#">ColorText</a> 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()`.



## 0.5.1.2.62 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	<i>p</i>	Pointer to a <a href="#">ColorText</a> .
----	----------	--

See also

[ColorText](#)

Referenced by ColorText\_free().

## 0.5.1.2.63 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	<i>text</i>	Text to colorize (a regular string).
in	...	ColorArgs for fore, back, and style, in any order.

Returns

An initialized [ColorText](#) struct.

See also

[ColorText](#)

0.5.1.2.64 `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	<i>ctext</i>	The <a href="#">ColorText</a> to inspect.
in	<i>carg</i>	The <a href="#">ColorArg</a> to look for.

Returns

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

See also

[ColorText](#)

0.5.1.2.65 `ColorText_has_args()`

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

Checks to see if a [ColorText](#) has any argument values set.

Parameters

in	<i>ctext</i>	A <a href="#">ColorText</a> to check.
----	--------------	---------------------------------------

Returns

true if .fore, .back, or .style is set to a non-empty [ColorArg](#), otherwise false.

See also

[ColorText](#)

0.5.1.2.66 `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	<i>ctext</i>	The <a href="#">ColorText</a> to check.
----	--------------	---

Returns

true if the [ColorText](#) is empty, otherwise false.

See also

[ColorText](#)  
[ColorText\\_empty](#)

0.5.1.2.67 `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	<i>p</i>	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()`.

## 0.5.1.2.68 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	<i>ctext</i>	<a href="#">ColorText</a> 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\(\)](#).

## 0.5.1.2.69 ColorText\_repr()

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

Allocate a string (char\*) representation for a [ColorText](#).

## Parameters

in	<i>ctext</i>	<a href="#">ColorText</a> to get the string representation for.
----	--------------	---

## Returns

Allocated string for the [ColorText](#).

## See also

[ColorText](#)

Referenced by [\\_colr\\_ptr\\_repr\(\)](#).

## 0.5.1.2.70 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	<i>ctext</i>	The <a href="#">ColorText</a> to set the justification method for.
in	<i>cjust</i>	The <a href="#">ColorJustify</a> struct to use.

## Returns

The same pointer that was given as *ctext*.

## See also

[ColorText](#)

## 0.5.1.2.71 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	<i>ctext</i>	A <a href="#">ColorText</a> to initialize with values.
in	<i>text</i>	Text to colorize (a regular string).
in	...	A <i>va_list</i> with <a href="#">ColorArgs</a> pointers for fore, back, and style, in any order.

## Returns

An initialized [ColorText](#) struct.

## See also

[ColorText](#)

## 0.5.1.2.72 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	<i>ctext</i>	<a href="#">ColorText</a> 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](#)

## 0.5.1.2.73 ColorText\_to\_str()

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

Stringifies a [ColorText](#) struct, creating a mix of escape codes and text.

## Parameters

in	<i>ctext</i>	<a href="#">ColorText</a> 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()`.

## 0.5.1.2.74 ColorType\_eq()

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

Compares two ColorTypes.

This is used to implement [colr\\_eq\(\)](#).

Parameters

in	<i>a</i>	The first ColorType to compare.
in	<i>b</i>	The second ColorType to compare.

Returns

true if they are equal, otherwise false.

See also

[ColorType](#)

## 0.5.1.2.75 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	<i>arg</i>	Color name to get the ColorType for.
----	------------	--------------------------------------

Return values

<i>ColorType</i>	value on success.
<i>TYPE_INVALID</i>	for invalid color names/strings.
<i>TYPE_INVALID_EXT_RANGE</i>	for ExtendedValues outside of 0-255.

Return values

<code>TYPE_INVALID_RGB_RANGE</code>	for rgb values outside of 0-255.
-------------------------------------	----------------------------------

See also

[ColorType](#)

#### 0.5.1.2.76 ColorType\_is\_invalid()

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

Check to see if a ColorType value is considered invalid.

Parameters

in	<i>type</i>	ColorType value to check.
----	-------------	---------------------------

Returns

true if the value is considered invalid, otherwise false.

See also

[ColorType](#)

#### 0.5.1.2.77 ColorType\_is\_valid()

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

Check to see if a ColorType value is considered valid.

Parameters

in	<i>type</i>	ColorType value to check.
----	-------------	---------------------------

Returns

true if the value is considered valid, otherwise false.



See also

[ColorType](#)

#### 0.5.1.2.78 ColorType\_repr()

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

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

Parameters

in	<i>type</i>	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](#)

#### 0.5.1.2.79 ColorType\_to\_str()

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

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

Parameters

in	<i>type</i>	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\(\)](#).

#### 0.5.1.2.80 [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](#)

#### 0.5.1.2.81 [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	<i>a</i>	First <a href="#">ColorValue</a> to compare.
in	<i>b</i>	Second <a href="#">ColorValue</a> to compare.

Returns

true if they are equal, otherwise false.

See also

[ColorValue](#)

Referenced by ColorArg\_eq().

#### 0.5.1.2.82 ColorValue\_example()

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

Create a string (char\*) representation of a [ColorValue](#) with a human-friendly type/name.

Parameters

in	<i>cval</i>	A <a href="#">ColorValue</a> 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().

#### 0.5.1.2.83 ColorValue\_from\_esc()

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

Convert an escape-code string (char\*) into a [ColorValue](#).

Parameters

in	<i>s</i>	An escape-code string to parse. <i>Must be null-terminated.</i>
----	----------	--

Returns

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

## Return values

<i>For</i>	invalid strings, the <code>.type</code> member can be one of: <ul style="list-style-type: none"> <li>• <code>TYPE_INVALID</code></li> <li>• <code>TYPE_INVALID_EXT_RANGE</code></li> <li>• <code>TYPE_INVALID_RGB_RANGE</code></li> </ul>
------------	---

## See also

[ColorValue](#)  
[ColorArg\\_from\\_esc](#)

Referenced by `ColorArg_from_esc()`.

0.5.1.2.84 `ColorValue_from_str()`

[ColorValue](#) `ColorValue_from_str (`  
`const char * s )`

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

## Parameters

<code>in</code>	<code>s</code>	A string to parse the color name from (can be an <a href="#">RGB</a> string).
-----------------	----------------	---

## Returns

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

## Return values

<i>For</i>	invalid strings, the <code>.type</code> member can be one of: <ul style="list-style-type: none"> <li>• <code>TYPE_INVALID</code></li> <li>• <code>TYPE_INVALID_EXT_RANGE</code></li> <li>• <code>TYPE_INVALID_RGB_RANGE</code></li> </ul>
------------	---

## See also

[ColorValue](#)

Referenced by `ColorArg_from_str()`.

## 0.5.1.2.85 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	<i>type</i>	A <a href="#">ColorType</a> value, to mark the type of <a href="#">ColorValue</a> .
in	<i>p</i>	A pointer to either a <a href="#">BasicValue</a> , <a href="#">ExtendedValue</a> , or a <a href="#">RGB</a> .

## 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\(\)](#).

## 0.5.1.2.86 ColorValue\_has\_BasicValue()

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

Checks to see if a [ColorValue](#) has a [BasicValue](#) set.

## Parameters

in	<i>cval</i>	<a href="#">ColorValue</a> to check.
in	<i>bval</i>	<a href="#">BasicValue</a> to look for.

## Returns

true if the [ColorValue](#) has the exact [BasicValue](#) set.

See also

[ColorValue](#)

#### 0.5.1.2.87 ColorValue\_has\_ExtendedValue()

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

Checks to see if a [ColorValue](#) has a [ExtendedValue](#) set.

Parameters

in	<i>cval</i>	<a href="#">ColorValue</a> to check.
in	<i>eval</i>	<a href="#">ExtendedValue</a> to look for.

Returns

true if the [ColorValue](#) has the exact [ExtendedValue](#) set.

See also

[ColorValue](#)

#### 0.5.1.2.88 ColorValue\_has\_RGB()

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

Checks to see if a [ColorValue](#) has a [RGB](#) value set.

Parameters

in	<i>cval</i>	<a href="#">ColorValue</a> to check.
in	<i>rgb</i>	<a href="#">RGB</a> value to look for.

Returns

true if the [ColorValue](#) has the exact [RGB](#) value set.

See also

[ColorValue](#)

## 0.5.1.2.89 ColorValue\_has\_StyleValue()

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

Checks to see if a [ColorValue](#) has a StyleValue set.

Parameters

in	<i>cval</i>	<a href="#">ColorValue</a> to check.
in	<i>sval</i>	StyleValue to look for.

Returns

true if the [ColorValue](#) has the exact StyleValue set.

See also

[ColorValue](#)

## 0.5.1.2.90 ColorValue\_is\_empty()

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

Checks to see if a [ColorValue](#) is an empty placeholder.

Parameters

in	<i>cval</i>	<a href="#">ColorValue</a> to check.
----	-------------	--------------------------------------

Returns

true if the [ColorValue](#) is "empty", otherwise false.

See also

[ColorValue](#)  
[ColorValue\\_empty](#)  
[ColorArg\\_empty](#)  
[ColorArg\\_is\\_empty](#)

## 0.5.1.2.91 ColorValue\_is\_invalid()

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

Checks to see if a [ColorValue](#) holds an invalid value.

## Parameters

in	<i>cval</i>	<a href="#">ColorValue</a> struct to check.
----	-------------	---

## Returns

true if the value is invalid, otherwise false.

## See also

[ColorValue](#)

Referenced by [ColorArg\\_from\\_esc\(\)](#).

0.5.1.2.92 [ColorValue\\_is\\_valid\(\)](#)

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

Checks to see if a [ColorValue](#) holds a valid value.

## Parameters

in	<i>cval</i>	<a href="#">ColorValue</a> struct to check.
----	-------------	---

## Returns

true if the value is valid, otherwise false.

## See also

[ColorValue](#)

0.5.1.2.93 [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	<i>type</i>	<a href="#">ArgType</a> (FORE, BACK, STYLE)
in	<i>cval</i>	<a href="#">ColorValue</a> 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()`.

0.5.1.2.94 `ColorValue_repr()`

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

Creates a string (`char*`) representation of a [ColorValue](#).

## Parameters

in	<i>cval</i>	A <a href="#">ColorValue</a> 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()`.

0.5.1.2.95 `ColorValue_to_esc()`

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

Converts a [ColorValue](#) into an escape code string (`char*`).

## Parameters

in	<i>type</i>	<code>ArgType</code> (FORE, BACK, STYLE) to build the escape code for.
in	<i>cval</i>	<a href="#">ColorValue</a> 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()`.

0.5.1.2.96 `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	<i>dest</i>	Destination string for the escape code string. <i>Must have room for the code type being used.</i>
in	<i>type</i>	<code>ArgType</code> ( <code>FORE</code> , <code>BACK</code> , <code>STYLE</code> ) to build the escape code for.
in	<i>cval</i>	<a href="#">ColorValue</a> 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()`.

0.5.1.2.97 `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	<i>match</i>	A <code>regmatch_t</code> to allocate for and copy.
----	--------------	---

## Returns

An allocated copy of the `regmatch_t`.

Referenced by `colr_re_matches()`.

0.5.1.2.98 `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	<i>s</i>	The string to append to. <i>Must have extra room for <code>CODE_RESET_ALL</code>. Must be null-terminated.</i>
----	----------	--

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

0.5.1.2.99 `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
<code>\'</code>	single quote
<code>\"</code>	double quote
<code>\?</code>	question mark
<code>\\</code>	backslash
<code>\a</code>	audible bell
<code>\b</code>	backspace
<code>\f</code>	form feed - new page
<code>\n</code>	line feed - new line
<code>\r</code>	carriage return
<code>\t</code>	horizontal tab
<code>\v</code>	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()`.

0.5.1.2.100 `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. Input <i>must be null-terminated</i> .

## Returns

true if c is found in s, otherwise false.

Referenced by `colr_str_chars_lcount()`, and `colr_str_lstrip_chars()`.

0.5.1.2.101 `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](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()`.

0.5.1.2.102 `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()`.

0.5.1.2.103 `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

Escape Sequence	Description Representation
<code>\b</code>	backspace
<code>\f</code>	form feed - new page
<code>\n</code>	line feed - new line
<code>\r</code>	carriage return
<code>\t</code>	horizontal tab
<code>\v</code>	vertical tab

#### Parameters

in	<i>c</i>	The character to check.
----	----------	-------------------------

#### Returns

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

Referenced by `colr_str_repr()`.

#### 0.5.1.2.104 `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	<i>marker</i>	A colr marker, like <code>COLORARG_MARKER</code> , <code>COLORTEXT_MARKER</code> , etc.
in	<i>p</i>	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()`.

## 0.5.1.2.105 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().

## 0.5.1.2.106 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	<i>matches</i>	A pointer to an array of regmatch_t pointers.
-----	----------------	---

Referenced by colr\_str\_replace\_re\_pat\_all().

## 0.5.1.2.107 colr\_join\_array()

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

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

## Parameters

in	<i>joinerp</i>	The joiner (any <a href="#">ColorArg</a> , <a href="#">ColorText</a> , or string (char*)).
in	<i>ps</i>	An array of pointers to <a href="#">ColorArgs</a> , <a href="#">ColorTexts</a> , 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](#).

0.5.1.2.108 `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	<i>joinerp</i>	The joiner (any <a href="#">ColorArg</a> , <a href="#">ColorText</a> , or string ( <code>char*</code> )).
in	<i>ps</i>	An array of pointers to <a href="#">ColorArgs</a> , <a href="#">ColorTexts</a> , or strings ( <code>char*</code> ). The array must have at least a length of <i>count</i> , unless a <code>NULL</code> element is placed at the end.
in	<i>count</i>	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()`.



0.5.1.2.109 `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	<code>s</code>	The string to check.
in	<code>length</code>	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

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

## See also

[`colr\_str\_mb\_len`](#)  
[`colr\_is\_valid\_mblen`](#)

Referenced by `_rainbow()`.

0.5.1.2.110 `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	<i>fp</i>	FILE pointer for output.
in	<i>info</i>	Info from printf about how to format the argument.
in	<i>args</i>	Argument list (with only 1 argument), containing a <a href="#">ColorArg</a> , <a href="#">ColorResult</a> , <a href="#">ColorText</a> , or string (char*) to format.

## Returns

The number of characters written.

Referenced by `colr_printf_register()`.

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

## Returns

The number of argument types set in `argtypes`.

Referenced by `colr_printf_register()`.

0.5.1.2.112 `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.

0.5.1.2.113 `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	<code>s</code>	The string to search.
in	<code>repattern</code>	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()`.

0.5.1.2.114 `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()`.

0.5.1.2.115 `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	<i>lst</i>	The string array to look in.
in	<i>s</i>	The string to look for.

## Returns

true if the string is found, otherwise false.

## Return values

<code>&lt;tt&gt;false&lt;/tt&gt;</code>	if <i>lst</i> is <code>NULL</code> or <i>s</i> is <code>NULL</code> .
---	---

Referenced by `colr_str_get_codes()`.

0.5.1.2.116 `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	<i>ps</i>	A pointer to an array of strings.
----	-----------	-----------------------------------

Referenced by `ColorArgs_from_str()`.

0.5.1.2.117 `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	<code>s</code>	The string to justify. Input <i>must be null-terminated</i> .
in	<code>width</code>	The overall width for the resulting string. If set to '0', the terminal width will be used from <a href="#">colr_term_size()</a> .
in	<code>padchar</code>	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()`.

0.5.1.2.118 `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	<code>s</code>	The string to examine. <i>Must be null-terminated.</i>
in	<code>c</code>	The character to count. <i>Must not be 0.</i>

## Returns

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

Referenced by `_rainbow()`.

0.5.1.2.119 `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	<code>s</code>	The string to examine. <i>Must be null-terminated.</i>
in	<code>c</code>	The character to count. <i>Must not be 0.</i>

## Returns

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

Referenced by `colr_str_lstrip_char()`.

0.5.1.2.120 `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	<code>s</code>	The string to examine. <i>Must be null-terminated.</i>
in	<code>chars</code>	The characters to count, in any order. <i>Must not be 0.</i>

## Returns

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

Referenced by `colr_str_lstrip_chars()`.

0.5.1.2.121 `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	<code>s</code>	A string to count the escape-codes for. <i>Must be null-terminated.</i>
----	----------------	--

## 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()`.

0.5.1.2.122 `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	<code>s</code>	A string to count the code-chars for. <i>Must be null-terminated.</i>
----	----------------	--

## Returns

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

0.5.1.2.123 `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. <i>Must have room for strlen(src) + 1 or length + 1.</i>
in	src	Source string to copy.
in	length	Maximum characters to copy. <i>This does not include the null-terminator.</i> Usually set to strlen(dest).

Returns

On success, a pointer to dest is returned.

#### 0.5.1.2.124 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. <i>Must be null-terminated.</i>
in	suffix	Suffix to check for. <i>Must be null-terminated.</i>

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().



0.5.1.2.125 `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	<code>s</code>	A string to get the escape-codes from. <i>Must be null-terminated.</i>
in	<code>unique</code>	Whether to only include <i>unique</i> 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

<i>If</i>	<code>s</code> 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.
<i>On</i>	success, there will be at least two pointers behind the return value. The last pointer is always NULL.

Referenced by `ColorArgs_from_str()`.

0.5.1.2.126 `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	<code>s</code>	The string to check. Can be NULL. Input <i>must be null-terminated.</i>
----	----------------	--

## Returns

true if the string has at least one escape code, otherwise false.

## See also

[colr\\_str\\_is\\_codes](#)

## 0.5.1.2.127 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. <i>Must be null-terminated.</i>
----	---	--

## 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()`.

## 0.5.1.2.128 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.

0.5.1.2.129 `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. Input <i>must be null-terminated</i> .
----	---	--

## Returns

true if the string is escape-codes only, otherwise false.

See also

[colr\\_str\\_has\\_codes](#)

0.5.1.2.130 `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. Input <i>must be null-terminated</i> .
----	---	--

## Returns

true if all characters are digits (0-9), otherwise false.

Referenced by `ExtendedValue_from_str()`.

0.5.1.2.131 `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	<code>s</code>	The string to justify. Input <i>must be null-terminated</i> .
in	<code>width</code>	The overall width for the resulting string. If set to '0', the terminal width will be used from <a href="#">colr_term_size()</a> .
in	<code>padchar</code>	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()`.

0.5.1.2.132 `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	<code>s</code>	The input string to convert to lower case.
----	----------------	--

## 0.5.1.2.133 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	<i>dest</i>	Destination char array. Must have room for <code>strlen(s) + 1</code> .
in	<i>s</i>	String to strip the character from.
in	<i>length</i>	Length of <i>s</i> , the input string.
in	<i>c</i>	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()`.

## 0.5.1.2.134 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	<i>s</i>	String to strip the character from.
in	<i>c</i>	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.*

0.5.1.2.135 `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	<code>s</code>	The string to strip. <i>s</i> <i>Must be null-terminated.</i>
in	<code>chars</code>	A string of characters to remove. Each will be removed from the start of the string. <i>chars</i> <i>Must be null-terminated.</i>

## 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.*

0.5.1.2.136 `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	<code>s</code>	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()`.

0.5.1.2.137 `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	<i>s</i>	String to get the length for. Input <i>must be null-terminated</i> .
----	----------	---

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()`.

0.5.1.2.138 `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	<i>s</i>	The string to operate on.
in	<i>target</i>	The string to replace.
in	<i>repl</i>	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()`.

#### 0.5.1.2.139 `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	<code>s</code>	The string to operate on.
in	<code>target</code>	The string to replace.
in	<code>repl</code>	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()`.

#### 0.5.1.2.140 `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	<i>s</i>	The string to operate on.
in	<i>target</i>	The string to replace.
in	<i>repl</i>	The <a href="#">ColorArg</a> to produce escape-codes to replace with. <a href="#">ColorArg_free()</a> 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](#)

## 0.5.1.2.141 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	<i>s</i>	The string to operate on.
in	<i>target</i>	The string to replace.
in	<i>repl</i>	The <a href="#">ColorResult</a> to produce escape-codes to replace with. <a href="#">ColorResult_free()</a> 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](#)

0.5.1.2.142 `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	<code>s</code>	The string to operate on.
in	<code>target</code>	The string to replace.
in	<code>repl</code>	The <a href="#">ColorText</a> to produce text/escape-codes to replace with. <a href="#">ColorText_free()</a> 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](#)

0.5.1.2.143 `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	<code>s</code>	The string to operate on.
in	<code>target</code>	The string to replace.
in	<code>repl</code>	The string to replace with.
in	<code>count</code>	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()`.

0.5.1.2.144 `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	<code>s</code>	The string to operate on.
in	<code>target</code>	The string to replace.
in	<code>repl</code>	The <a href="#">ColorArg</a> to produce escape-codes to replace with. <a href="#">ColorArg_free()</a> 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](#)

0.5.1.2.145 `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	<i>s</i>	The string to operate on.
in	<i>target</i>	The string to replace.
in	<i>repl</i>	The <a href="#">ColorResult</a> to produce escape-codes to replace with. <a href="#">ColorResult_free()</a> 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](#)

## 0.5.1.2.146 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	<i>s</i>	The string to operate on.
in	<i>target</i>	The string to replace.
in	<i>repl</i>	The <a href="#">ColorText</a> to produce text/escape-codes to replace with. <a href="#">ColorText_free()</a> 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](#)

0.5.1.2.147 `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	<code>s</code>	The string to operate on.
in	<code>pattern</code>	The regex match object to find text to replace.
in	<code>repl</code>	The string to replace with.
in	<code>re_flags</code>	Flags for <code>regcomp()</code> . <code>REG_EXTENDED</code> 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()`.

0.5.1.2.148 `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	<i>s</i>	The string to operate on.
in	<i>pattern</i>	The regex match object to find text to replace.
in	<i>repl</i>	The string to replace with.
in	<i>re_flags</i>	Flags for <code>regcomp()</code> . <code>REG_EXTENDED</code> 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()`.

0.5.1.2.149 `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	<i>s</i>	The string to operate on.
in	<i>pattern</i>	The regex pattern to compile.
in	<i>repl</i>	The <a href="#">ColorArg</a> to produce escape-codes to replace with. <a href="#">ColorArg_free()</a> is called after the replacement is done.
in	<i>re_flags</i>	Flags for <code>regcomp()</code> . <code>REG_EXTENDED</code> 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](#)

## 0.5.1.2.150 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	<i>s</i>	The string to operate on.
in	<i>pattern</i>	The regex match object to find text to replace.
in	<i>repl</i>	The <a href="#">ColorResult</a> to produce escape-codes to replace with. <a href="#">ColorResult_free()</a> is called after the replacement is done.
in	<i>re_flags</i>	Flags for <code>regcomp()</code> . 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](#)

0.5.1.2.151 `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	<code>s</code>	The string to operate on.
in	<code>pattern</code>	The regex match object to find text to replace.
in	<code>repl</code>	The <code>ColorText</code> to produce text/escape-codes to replace with. <code>ColorText_free()</code> is called after the replacement is done.
in	<code>re_flags</code>	Flags for <code>regcomp()</code> . <code>REG_EXTENDED</code> 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`

0.5.1.2.152 `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	<code>s</code>	The string to operate on.
in	<code>pattern</code>	The regex pattern to compile.
in	<code>repl</code>	The <code>ColorArg</code> to produce escape-codes to replace with. <code>ColorArg_free()</code> is called after the replacement is done.
in	<code>re_flags</code>	Flags for <code>regcomp()</code> . <code>REG_EXTENDED</code> 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](#)

## 0.5.1.2.153 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	<i>s</i>	The string to operate on.
in	<i>pattern</i>	The regex match object to find text to replace.
in	<i>repl</i>	The <a href="#">ColorResult</a> to produce escape-codes to replace with. <a href="#">ColorResult_free()</a> is called after the replacement is done.
in	<i>re_flags</i>	Flags for <code>regcomp()</code> . 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](#)

0.5.1.2.154 `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	<code>s</code>	The string to operate on.
in	<code>pattern</code>	The regex match object to find text to replace.
in	<code>repl</code>	The <code>ColorText</code> to produce text/escape-codes to replace with. <code>ColorText_free()</code> is called after the replacement is done.
in	<code>re_flags</code>	Flags for <code>regcomp()</code> . <code>REG_EXTENDED</code> 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](#)

0.5.1.2.155 `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	<code>s</code>	The string to operate on.
in	<code>match</code>	The regex match object to find text to replace.
in	<code>repl</code>	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\(\)](#).

## 0.5.1.2.156 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	<i>s</i>	The string to operate on.
in	<i>match</i>	The regex match object to find text to replace.
in	<i>repl</i>	The <a href="#">ColorArg</a> to produce escape-codes to replace with. <a href="#">ColorArg_free()</a> 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](#)

0.5.1.2.157 `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	<code>s</code>	The string to operate on.
in	<code>match</code>	The regex match object to find text to replace.
in	<code>repl</code>	The <a href="#">ColorResult</a> to produce escape-codes to replace with. <a href="#">ColorResult_free()</a> 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](#)

0.5.1.2.158 `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	<code>s</code>	The string to operate on.
in	<code>match</code>	The regex match object to find text to replace.
in	<code>repl</code>	The <a href="#">ColorText</a> to produce text/escape-codes to replace with. <a href="#">ColorText_free()</a> 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](#)

## 0.5.1.2.159 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	<i>ref</i>	The string to use for offset references. Can be <i>target</i> . Set this to the source string if <i>target</i> has not been filled yet. If <i>target</i> has been filled, you may use <i>target</i> for both <i>ref</i> and <i>target</i> .
out	<i>target</i>	The string to modify. Must have room for the resulting string.
in	<i>match</i>	The regex match object to find text to replace.
in	<i>repl</i>	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()*.

0.5.1.2.160 `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	<code>s</code>	The string to operate on.
in	<code>matches</code>	Regex match objects to find text to replace. The array must have <code>NULL</code> as the last member.
in	<code>repl</code>	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_ColorResult()`, `colr_str_replace_re_matches_ColorText()`, and `colr_str_replace_re_pat_all()`.

0.5.1.2.161 `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	<code>s</code>	The string to operate on.
in	<code>matches</code>	The regex match objects to find text to replace.
in	<code>repl</code>	The <a href="#">ColorArg</a> to produce escape-codes to replace with. <a href="#">ColorArg_free()</a> 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](#)

0.5.1.2.162 `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	<code>s</code>	The string to operate on.
in	<code>matches</code>	The regex match objects to find text to replace.
in	<code>repl</code>	The <a href="#">ColorResult</a> to produce escape-codes to replace with. <a href="#">ColorResult_free()</a> 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](#)

0.5.1.2.163 `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	<code>s</code>	The string to operate on.
in	<code>matches</code>	The regex match objects to find text to replace.
in	<code>repl</code>	The <a href="#">ColorText</a> to produce text/escape-codes to replace with. <a href="#">ColorText_free()</a> 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](#)

#### 0.5.1.2.164 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	<code>s</code>	The string to operate on.
in	<code>repattern</code>	The regex pattern to match ( <code>regex_t*</code> ).
in	<code>repl</code>	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()`.

## 0.5.1.2.165 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	<i>s</i>	The string to operate on.
in	<i>repattern</i>	The regex pattern to match (regex_t*).
in	<i>repl</i>	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()`.

0.5.1.2.166 `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	<code>s</code>	The string to operate on.
in	<code>repattern</code>	The regex pattern to match ( <code>regex_t*</code> ).
in	<code>repl</code>	The <a href="#">ColorArg</a> to produce escape-codes to replace with. <a href="#">ColorArg_free()</a> 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](#)

0.5.1.2.167 `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	<code>s</code>	The string to operate on.
in	<code>repattern</code>	The regex pattern to match ( <code>regex_t*</code> ).
in	<code>repl</code>	The <a href="#">ColorResult</a> to produce escape-codes to replace with. <a href="#">ColorResult_free()</a> 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](#)

## 0.5.1.2.168 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	<i>s</i>	The string to operate on.
in	<i>repattern</i>	The regex pattern to match (regex_t*).
in	<i>repl</i>	The <a href="#">ColorText</a> to produce text/escape-codes to replace with. <a href="#">ColorText_free()</a> 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](#)

## 0.5.1.2.169 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	<i>s</i>	The string to operate on.
in	<i>repattern</i>	The regex pattern to match ( <code>regex_t*</code> ).
in	<i>repl</i>	The <a href="#">ColorArg</a> to produce escape-codes to replace with. <a href="#">ColorArg_free()</a> 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](#)

## 0.5.1.2.170 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	<i>s</i>	The string to operate on.
in	<i>repattern</i>	The regex pattern to match ( <code>regex_t*</code> ).
in	<i>repl</i>	The <a href="#">ColorResult</a> to produce escape-codes to replace with. <a href="#">ColorResult_free()</a> 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](#)

0.5.1.2.171 `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	<code>s</code>	The string to operate on.
in	<code>repattern</code>	The regex pattern to match ( <code>regex_t*</code> ).
in	<code>repl</code>	The <a href="#">ColorText</a> to produce text/escape-codes to replace with. <a href="#">ColorText_free()</a> 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](#)

0.5.1.2.172 `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	<code>s</code>	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()`.

0.5.1.2.173 `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	<code>s</code>	The string to justify. Input <i>must be null-terminated</i> .
in	<code>width</code>	The overall width for the resulting string. If set to '0', the terminal width will be used from <a href="#">colr_term_size()</a> .
in	<code>padchar</code>	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()`.

## 0.5.1.2.174 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	<i>s</i>	The string to check.
in	<i>prefix</i>	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.

0.5.1.2.175 `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	<i>s</i>	The string to strip escape codes from. Input <i>must be null-terminated</i> .
----	----------	--

## 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()`.

0.5.1.2.176 `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. <i>Must be null-terminated.</i>
----	---	---

## 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().

## 0.5.1.2.177 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().

## 0.5.1.2.178 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.

0.5.1.2.179 `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()`.

0.5.1.2.180 `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()`.

0.5.1.2.181 `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()`.

0.5.1.2.182 `ExtendedValue_eq()`

```
bool ExtendedValue_eq (
    ExtendedValue a,
    ExtendedValue b )
```

Compares two `ExtendedValues`.

This is used to implement `colr_eq()`.

Parameters

in	<i>a</i>	The first <code>ExtendedValue</code> to compare.
in	<i>b</i>	The second <code>ExtendedValue</code> to compare.

Returns

true if they are equal, otherwise false.

See also

[ExtendedValue](#)

0.5.1.2.183 `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	<i>bval</i>	BasicValue to convert.
----	-------------	------------------------

## Returns

An ExtendedValue 0–15 on success, otherwise EXT\_INVALID.

## See also

[ExtendedValue](#)

## 0.5.1.2.184 ExtendedValue\_from\_esc()

```
int ExtendedValue_from_esc (
    const char * s )
```

Convert an escape-code string (char\*) to an ExtendedValue.

## Parameters

in	<i>s</i>	Escape-code string. <i>Must be null-terminated.</i>
----	----------	--

## Return values

<i>An</i>	integer in the range 0–255 on success.
<i>EXT_INVALID</i>	on error (or if <i>s</i> is NULL).
<i>EXT_INVALID_RANGE</i>	if the code number was outside of the range 0–255.

## See also

[ExtendedValue](#)

## 0.5.1.2.185 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	<i>hexstr</i>	Hex string to convert.
----	---------------	------------------------

## Returns

A value between 0 and 255 on success.

## Return values

<i>COLOR_INVALID</i>	on error or bad values.
----------------------	-------------------------

## See also

[ExtendedValue](#)

Referenced by `ExtendedValue_from_hex_default()`, and `ExtendedValue_from_str()`.

0.5.1.2.186 `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	<i>hexstr</i>	Hex string to convert.
in	<i>default_value</i>	<code>ExtendedValue</code> to use for bad hex strings.

## Returns

An `ExtendedValue` on success, or `default_value` on error.

See also

[ExtendedValue](#)  
[ExtendedValue\\_from\\_hex](#)

#### 0.5.1.2.187 ExtendedValue\_from\_RGB()

```
ExtendedValue ExtendedValue_from_RGB (
    RGB rgb )
```

Convert an [RGB](#) value into the closest matching [ExtendedValue](#).

Parameters

in	<i>rgb</i>	<a href="#">RGB</a> 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\(\)](#).

#### 0.5.1.2.188 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	<i>arg</i>	Color name to find the <a href="#">ExtendedValue</a> for.
----	------------	---

## Returns

A value between 0 and 255 on success.

## Return values

<i>EXT_INVALID</i>	on error or bad values.
<i>EXT_INVALID_RANGE</i>	if the number was outside of the range 0-255.

## See also

[ExtendedValue](#)

## 0.5.1.2.189 ExtendedValue\_is\_invalid()

```
bool ExtendedValue_is_invalid (  
    int eval )
```

Determines whether an integer is an invalid ExtendedValue.

## Parameters

in	<i>eval</i>	A number to check.
----	-------------	--------------------

## Returns

true if the value is considered invalid, otherwise false.

## See also

[ExtendedValue](#)

## 0.5.1.2.190 ExtendedValue\_is\_valid()

```
bool ExtendedValue_is_valid (  
    int eval )
```

Determines whether an integer is a valid ExtendedValue.

## Parameters

in	<i>eval</i>	A number to check.
----	-------------	--------------------

## Returns

true if the value is considered valid, otherwise false.

## See also

[ExtendedValue](#)

## 0.5.1.2.191 ExtendedValue\_repr()

```
char* ExtendedValue_repr (
    int eval )
```

Creates a string (char\*) representation of a ExtendedValue.

## Parameters

in	<i>eval</i>	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](#)

## 0.5.1.2.192 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	<i>eval</i>	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](#)

#### 0.5.1.2.193 `format_bg()`

```
void format_bg (
    char * out,
    BasicValue value )
```

Create an escape code for a background color.

Parameters

out	<i>out</i>	Memory allocated for the escape code string. <i>Must have enough room for CODEX_LEN.</i>
in	<i>value</i>	<a href="#">BasicValue</a> value to use for background.

#### 0.5.1.2.194 `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	<i>out</i>	Memory allocated for the escape code string. <i>Must have enough room for CODE_RGB_LEN.</i>
in	<i>rgb</i>	<a href="#">RGB</a> struct to get red, blue, and green values from.

Referenced by `_rainbow()`, and `rainbow_bg()`.

#### 0.5.1.2.195 `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	<i>out</i>	Memory allocated for the escape code string.
in	<i>rgb</i>	Pointer to an <a href="#">RGB</a> struct.

Referenced by `_rainbow()`, and `rainbow_bg_term()`.

0.5.1.2.196 `format_bgx()`

```
void format_bgx (
    char * out,
    unsigned char num )
```

Create an escape code for an extended background color.

## Parameters

out	<i>out</i>	Memory allocated for the escape code string. <i>Must have enough room for CODEX_LEN.</i>
in	<i>num</i>	Value to use for background.

Referenced by `format_bg_RGB_term()`.

0.5.1.2.197 `format_fg()`

```
void format_fg (
    char * out,
    BasicValue value )
```

Create an escape code for a fore color.

## Parameters

out	<i>out</i>	Memory allocated for the escape code string. <i>Must have enough room for CODEX_LEN.</i>
in	<i>value</i>	<a href="#">BasicValue</a> value to use for fore.

0.5.1.2.198 `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	<i>out</i>	Memory allocated for the escape code string.
in	<i>rgb</i>	Pointer to an <a href="#">RGB</a> struct.

Referenced by rainbow\_fg().

## 0.5.1.2.199 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	<i>out</i>	Memory allocated for the escape code string.
in	<i>rgb</i>	Pointer to an <a href="#">RGB</a> struct.

Referenced by rainbow\_fg\_term().

## 0.5.1.2.200 format\_fgx()

```
void format_fgx (  
    char * out,  
    unsigned char num )
```

Create an escape code for an extended fore color.

## Parameters

out	<i>out</i>	Memory allocated for the escape code string. <i>Must have enough room for CODEX_LEN.</i>
in	<i>num</i>	Value to use for fore.

Referenced by format\_fg\_RGB\_term().

## 0.5.1.2.201 format\_style()

```
void format_style (  
    char * out,  
    StyleValue style )
```

Create an escape code for a style.

## Parameters

out	<i>out</i>	Memory allocated for the escape code string. <i>Must have enough room for STYLE_LEN.</i>
in	<i>style</i>	StyleValue value to use for style.

## 0.5.1.2.202 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	<i>s</i>	The string to colorize. Input <i>must be null-terminated.</i>
in	<i>freq</i>	Frequency ("tightness") for the colors.
in	<i>offset</i>	Starting offset in the rainbow.
in	<i>spread</i>	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.*

## 0.5.1.2.203 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	<i>s</i>	The string to colorize. Input <i>must be null-terminated</i> .
in	<i>freq</i>	Frequency ("tightness") for the colors.
in	<i>offset</i>	Starting offset in the rainbow.
in	<i>spread</i>	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.*

## 0.5.1.2.204 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	<i>s</i>	The string to colorize. Input <i>must be null-terminated</i> .
in	<i>freq</i>	Frequency ("tightness") for the colors.
in	<i>offset</i>	Starting offset in the rainbow.
in	<i>spread</i>	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.*

## 0.5.1.2.205 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	<i>s</i>	The string to colorize. Input <i>must be null-terminated</i> .
in	<i>freq</i>	Frequency ("tightness") for the colors.
in	<i>offset</i>	Starting offset in the rainbow.
in	<i>spread</i>	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.*

## 0.5.1.2.206 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	<i>freq</i>	Frequency ("tightness") of the colors.
in	<i>offset</i>	Starting offset in the rainbow.

## Returns

An [RGB](#) value with the next "step" in the "rainbow".

Referenced by `_rainbow()`.



## 0.5.1.2.207 RGB\_average()

```
unsigned char RGB_average (
    RGB rgb )
```

Return the average for an [RGB](#) value.

This is also it's "grayscale" value.

Parameters

in	<i>rgb</i>	The <a href="#">RGB</a> value to get the average for.
----	------------	---

Returns

A value between 0–255.

See also

[RGB](#)

Referenced by RGB\_grayscale().

## 0.5.1.2.208 RGB\_eq()

```
bool RGB_eq (
    RGB a,
    RGB b )
```

Compare two [RGB](#) structs.

Parameters

in	<i>a</i>	First <a href="#">RGB</a> value to check.
in	<i>b</i>	Second <a href="#">RGB</a> 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().

## 0.5.1.2.209 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	<i>bval</i>	A BasicValue to get the <a href="#">RGB</a> value for.
----	-------------	--

## Returns

An [RGB](#) value that matches the BasicValue's color.

## See also

[RGB](#)

## 0.5.1.2.210 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	<i>s</i>	Escape-code string. <i>Must be null-terminated.</i>
out	<i>rgb</i>	Pointer to an <a href="#">RGB</a> struct to fill in the values for.

## Return values

<code>&lt;tt&gt;0&lt;/tt&gt;</code>	on success, with <code>rgb</code> filled with values.
<code>COLOR_INVALID</code>	on error (or if <code>s</code> is NULL).
<code>COLOR_INVALID_RANGE</code>	if any code numbers were outside of the range 0–255.

## See also

[RGB](#)

## 0.5.1.2.211 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	<i>eval</i>	An ExtendedValue to get the RGB value for.
----	-------------	--

Returns

An RGB value from `ext2rgb_map[]`.

See also

RGB

## 0.5.1.2.212 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	<i>hexstr</i>	String to check for hex values. Input <i>must be null-terminated</i> .
out	<i>rgb</i>	Pointer to an RGB struct to fill in the values for.

Return values

0	on success, with <i>rgb</i> filled with the values.
<i>COLOR_INVALID</i>	for non-hex strings.

See also

[RGB](#)

Referenced by `ExtendedValue_from_hex()`, `RGB_from_hex_default()`, and `RGB_from_str()`.

#### 0.5.1.2.213 `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	<i>hexstr</i>	String to check for <a href="#">RGB</a> values. Input <i>must be null-terminated</i> .
out	<i>default_value</i>	An <a href="#">RGB</a> value to use when errors occur.

Returns

A valid [RGB](#) value on success, or `default_value` on error.

See also

[RGB](#)  
[hex](#)

#### 0.5.1.2.214 `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	<i>arg</i>	String to check for <a href="#">RGB</a> values. Input <i>must be null-terminated</i> .
out	<i>rgb</i>	Pointer to an <a href="#">RGB</a> struct to fill in the values for.

## Return values

0	on success, with <code>rgb</code> filled with the values.
<code>COLOR_INVALID</code>	for non-rgb strings.
<code>COLOR_INVALID_RANGE</code>	for rgb values outside of 0-255.

See also

[RGB](#)

0.5.1.2.215 `RGB_grayscale()`

```
RGB RGB_grayscale (  
    RGB rgb )
```

Return a grayscale version of an [RGB](#) value.

## Parameters

in	<i>rgb</i>	The <a href="#">RGB</a> value to convert.
----	------------	---

## Returns

A grayscale [RGB](#) value.

See also

[RGB](#)

## 0.5.1.2.216 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	<i>rgb</i>	The RGB value to invert.
----	------------	--------------------------

## Returns

An "inverted" RGB value.

## See also

RGB

## 0.5.1.2.217 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	<i>rgb</i>	The RGB value to convert.
----	------------	---------------------------

## Returns

Either `rgb(1, 1, 1)` or `rgb(255, 255, 255)`.

## See also

RGB

## 0.5.1.2.218 RGB\_repr()

```
char* RGB_repr (  
    RGB rgb )
```

Creates a string (char\*) representation for an RGB value.

Allocates memory for the string representation.

## Parameters

in	<i>rgb</i>	<a href="#">RGB</a> struct to get the representation for.
----	------------	---

## Returns

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

## See also

[RGB](#)

0.5.1.2.219 `RGB_to_hex()`

```
char* RGB_to_hex (
    RGB rgb )
```

Converts an [RGB](#) value into a hex string (char\*).

## Parameters

in	<i>rgb</i>	<a href="#">RGB</a> 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](#)

0.5.1.2.220 `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	<i>rgb</i>	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](#)

## 0.5.1.2.221 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	<i>rgb</i>	RGB to convert.
----	------------	-----------------

## Returns

A new [RGB](#) with values close to a terminal code color.

## See also

[RGB](#)

Referenced by `ExtendedValue_from_RGB()`.

## 0.5.1.2.222 StyleValue\_eq()

```
bool StyleValue_eq (
    StyleValue a,
    StyleValue b )
```

Compares two StyleValues.

This is used to implement `colr_eq()`.



## Parameters

in	<i>a</i>	The first StyleValue to compare.
in	<i>b</i>	The second StyleValue to compare.

## Returns

true if they are equal, otherwise false.

See also

[StyleValue](#)

## 0.5.1.2.223 StyleValue\_from\_esc()

```
StyleValue StyleValue_from_esc (  
    const char * s )
```

Convert an escape-code string (char\*) to an actual StyleValue enum value.

## Parameters

in	<i>s</i>	Escape-code string. <i>Must be null-terminated.</i>
----	----------	--

## Return values

<i>StyleValue</i>	value on success.
<i>STYLE_INVALID</i>	on error (or if <i>s</i> is NULL).
<i>STYLE_INVALID_RANGE</i>	if the code number was outside of the range 0–255.

See also

[StyleValue](#)

## 0.5.1.2.224 StyleValue\_from\_str()

```
StyleValue StyleValue_from_str (  
    const char * arg )
```

Convert a named argument to actual StyleValue enum value.

## Parameters

in	<i>arg</i>	Style name to convert into a StyleValue.
----	------------	--

## Returns

A usable StyleValue value on success, or STYLE\_INVALID on error.

## See also

[StyleValue](#)

## 0.5.1.2.225 StyleValue\_is\_invalid()

```
bool StyleValue_is_invalid (  
    StyleValue sval )
```

Determines whether a StyleValue is invalid.

## Parameters

in	<i>sval</i>	A StyleValue to check.
----	-------------	------------------------

## Returns

true if the value is considered invalid, otherwise false.

## See also

[StyleValue](#)

## 0.5.1.2.226 StyleValue\_is\_valid()

```
bool StyleValue_is_valid (  
    StyleValue sval )
```

Determines whether a StyleValue is valid.

## Parameters

in	<i>sval</i>	A StyleValue to check.
----	-------------	------------------------

## Returns

true if the value is considered valid, otherwise false.

## See also

[StyleValue](#)

## 0.5.1.2.227 StyleValue\_repr()

```
char* StyleValue_repr (
    StyleValue sval )
```

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

## Parameters

in	<i>sval</i>	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](#)

## 0.5.1.2.228 StyleValue\_to\_str()

```
char* StyleValue_to_str (
    StyleValue sval )
```

Create a human-friendly string (char\*) representation for a StyleValue.

## Parameters

in	<i>sval</i>	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](#)

#### 0.5.1.2.229 TermSize\_repr()

```
char* TermSize_repr (
    TermSize ts )
```

Create a string (char\*) representation for a [TermSize](#).

Parameters

in	ts	<a href="#">TermSize</a> 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](#)

### 0.5.1.3 Variable Documentation

#### 0.5.1.3.1 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\(\)](#).

#### 0.5.1.3.2 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()`.

#### 0.5.1.3.3 ext2rgb\_map

```
const RGB ext2rgb_map[]
```

A map from ExtendedValue (256-color) to [RGB](#) value, where the index 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](#)

#### 0.5.1.3.4 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\(\)](#).

## 0.5.1.3.5 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()`.

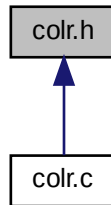
## 0.5.2 colr.h File Reference

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 *colname\_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 colored 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__, _ColrLastArg↵  
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, [Color↵Text](#) \*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, [Color↵Result](#) \*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, [Color↵Arg](#) \*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, [Color↵Arg](#) \*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_fg` (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.*

## 0.5.2.1 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](#).

## 0.5.2.2 Data Structure Documentation

### 0.5.2.2.1 struct BasicInfo

Holds a known color name and it's BasicValue.

This is used for the basic\_names array in [colr.c](#).

#### Data Fields

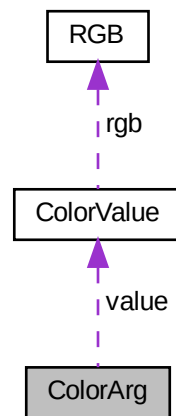
char *	name	
<a href="#">BasicValue</a>	value	

### 0.5.2.2.2 struct ColorArg

Holds an ArgType, and a [ColorValue](#).



Collaboration diagram for ColorArg:



Data Fields

<code>uint32_t</code>	marker	A marker used to inspect void pointers and determine if they are ColorArgs.
<a href="#">ArgType</a>	type	Fore, back, style, invalid.
<a href="#">ColorValue</a>	value	Color type and value.

#### 0.5.2.2.3 struct ColorJustify

Holds a string justification method, width, and padding character for ColorTexts.

Data Fields

<code>uint32_t</code>	marker	A marker used to inspect void pointers and determine if they are ColorJustifys.
<a href="#">ColorJustifyMethod</a>	method	The justification method, can be JUST_NONE.
<code>char</code>	padchar	The desired padding character, or 0 to use the default (" ").
<code>int</code>	width	The desired width for the final string, or 0 to use <a href="#">colr_term_size()</a> .

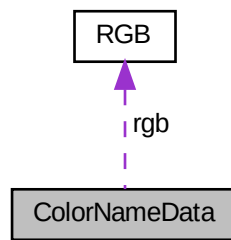
#### 0.5.2.2.4 struct ColorNameData

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

<a href="#">ExtendedValue</a>	ext	ExtendedValue (256-colors) for the color.
char *	name	The known name of the color.
<a href="#">RGB</a>	rgb	<a href="#">RGB</a> (TrueColor) for the color.

0.5.2.2.5 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 <a href="#">ColorResult</a> is initialized with a string ( <a href="#">ColorResult_new()</a> ). 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.

0.5.2.2.6 union ColorStructMarker

Breaks down Colr struct markers, such as COLORARG\_MARKER, into individual bytes.

Data Fields

struct <a href="#">ColorStructMarker</a>	bytes	Individual bytes that make up the marker.
uint32_t	marker	The actual uint32_t marker value.

## 0.5.2.2.7 struct ColorStructMarker.bytes

Individual bytes that make up the marker.

Data Fields

uint8_t	b1	
uint8_t	b2	
uint8_t	b3	
uint8_t	b4	

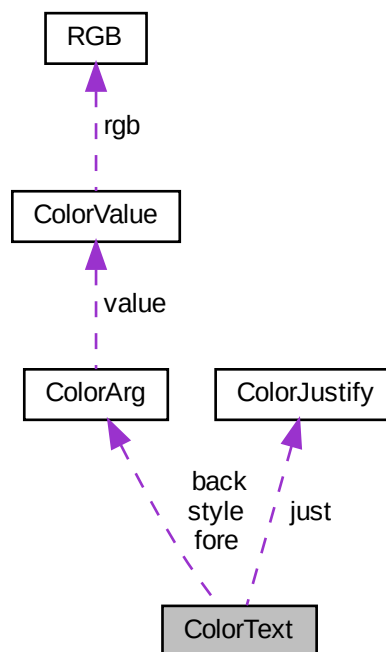
## 0.5.2.2.8 struct ColorText

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

<a href="#">ColorArg</a> *	back	<a href="#">ColorArg</a> for back color. Can be NULL.
----------------------------	------	---

## Data Fields

<a href="#">ColorArg</a> *	fore	<a href="#">ColorArg</a> for fore color. Can be NULL.
<a href="#">ColorJustify</a>	just	<a href="#">ColorJustify</a> info, set to JUST_NONE by default.
uint32_t	marker	A marker used to inspect void pointers and determine if they are ColorTexts.
<a href="#">ColorArg</a> *	style	<a href="#">ColorArg</a> for style value. Can be NULL.
char *	text	Text to colorize.

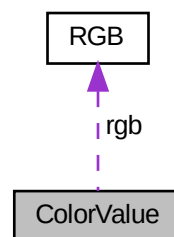
## 0.5.2.2.9 struct ColorValue

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

<a href="#">BasicValue</a>	basic	
<a href="#">ExtendedValue</a>	ext	
<a href="#">RGB</a>	rgb	
<a href="#">StyleValue</a>	style	
<a href="#">ColorType</a>	type	

## 0.5.2.2.10 struct ExtendedInfo

Holds a known color name and it's `ExtendedValue`.

This is used for the `basic_names` array in [color.c](#).

Data Fields

char *	name	
<a href="#">ExtendedValue</a>	value	

#### 0.5.2.2.11 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.

#### 0.5.2.2.12 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	
<a href="#">StyleValue</a>	value	

#### 0.5.2.2.13 struct TermSize

Holds a terminal size, usually retrieved with [colr\\_term\\_size\(\)](#).

Data Fields

unsigned short	columns	
unsigned short	rows	

### 0.5.2.3 Macro Definition Documentation

#### 0.5.2.3.1 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.*

## 0.5.2.3.2 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.*

## 0.5.2.3.3 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.*

## 0.5.2.3.4 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.*

## 0.5.2.3.5 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	<i>retval</i>	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()`.

## 0.5.2.3.6 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	<i>x</i>	A BasicValue, ExtendedValue, or <code>RGB</code> 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`.

## 0.5.2.3.7 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), <code>RGB</code> 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](#)

## 0.5.2.3.8 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 <code>RGB</code> struct.
----	---	--



## Returns

An allocated string.  
*You must `free()` the memory allocated by this function.*

## See also

[back](#)  
[back\\_arg](#)

0.5.2.3.9 `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 <a href="#">RGB</a> value.
----	---	--

## Returns

A stack-allocated escape code string.

## See also

[back\\_str\\_static](#)  
[style\\_str\\_static](#)  
[format\\_fg](#)  
[format\\_bg](#)

## 0.5.2.3.10 basic

```
#define basic(  
    x ) ((BasicValue)(x))
```

Casts to BasicValue.

## Parameters

in	<i>x</i>	Value to case to BasicValue.
----	----------	------------------------------

## Returns

A BasicValue.

## See also

[fore](#)  
[back](#)  
[colr](#)  
[Colr](#)

## 0.5.2.3.11 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	<i>x</i>	An enum to convert to boolean.
----	----------	--------------------------------

Return values

<i>true</i>	if the value is considered valid, or non-empty.
<i>false</i>	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().

#### 0.5.2.3.12 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  $(\text{CODE\_RGB\_LEN} * 2) + \text{STYLE\_LEN}$  since rgb codes are the longest).

Examples:

[colr\\_printf\\_example.c](#).

#### 0.5.2.3.13 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().

#### 0.5.2.3.14 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.

#### 0.5.2.3.15 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.

## 0.5.2.3.16 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.

## 0.5.2.3.17 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	<i>type</i>	ArgType (FORE, BACK, STYLE) to build the <a href="#">ColorArg</a> .
in	<i>x</i>	BasicValue, Extended (unsigned char). or <a href="#">RGB</a> value.

## Returns

`ColorArg_from_value(type, [appropriate type], x)`

## 0.5.2.3.18 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 colored must account for this.

## 0.5.2.3.19 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](#)

## 0.5.2.3.20 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](#)

## 0.5.2.3.21 COLOR\_RGB\_LEN

```
#define COLOR_RGB_LEN 26
```

Maximum length in chars added to a rgb colored string.

Should be `CODE_RGB_LEN + STYLE_LEN` Allocating for a string that will be colored with rgb values must account for this.

## 0.5.2.3.22 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 <a href="#">RGB</a> value.
----	---	---

## Returns

`ColorValue_from_value([appropriate type], x)`

## 0.5.2.3.23 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()`.

## 0.5.2.3.24 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	<i>cval</i>	The <a href="#">ColorValue</a> to check.
in	<i>val</i>	A BasicValue, ExtendedValue, StyleValue, or <a href="#">RGB</a> 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](#)

## 0.5.2.3.25 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	<i>text</i>	String to colorize/style.
in	...	No more than 3 <a href="#">ColorArg</a> 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](#).



## 0.5.2.3.26 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	<i>text</i>	String to colorize/style.
in	...	No more than 3 <a href="#">ColorArg</a> pointers for fore, back, and style in any order. <i>colr_free()</i> will be called on any <a href="#">ColorArg</a> , <a href="#">ColorResult</a> , or <a href="#">ColorText</a> 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](#).

## 0.5.2.3.27 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.

## 0.5.2.3.28 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'. <a href="#">colr_free()</a> will be called on any <a href="#">ColorArg</a> , <a href="#">ColorResult</a> , or <a href="#">ColorText</a> pointer passed to this function.
----	-----	--

## Returns

Same as `asprintf`.

## Examples:

[colr\\_printf\\_example.c](#).

## 0.5.2.3.29 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 <code>colr_cat()</code> , to concatenate. <i><code>colr_free()</code> will be called on any <code>ColorArg</code>, <code>ColorResult</code>, or <code>ColorText</code> pointer passed to this function.</i>
----	-----	--

## 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`.

0.5.2.3.30 `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 <code>ColorArg</code> pointers, <code>ColorResult</code> pointers, <code>ColorText</code> pointers, or strings to join. <i><code>colr_free()</code> will be called on any <code>ColorArg</code>, <code>ColorResult</code>, or <code>ColorText</code> pointer passed to this function.</i>
----	-----	--

## 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](#).

#### 0.5.2.3.31 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 it uses space as the default character.

Parameters

in	<i>text</i>	Text to colorize.
in	<i>justwidth</i>	Width for justification.
in	...	Fore, back, or style ColorArgs for <a href="#">Colr()</a> .

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](#).

#### 0.5.2.3.32 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	<i>text</i>	Text to colorize.
in	<i>justwidth</i>	Width for justification.
in	<i>c</i>	The character to pad with.
in	...	Fore, back, or style ColorArgs for <a href="#">Colr()</a> .

## 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](#)

## 0.5.2.3.33 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	<i>a</i>	First supported type to compare.
in	<i>b</i>	Second supported type to compare.

## Returns

true if the values are equal, otherwise false.

## 0.5.2.3.34 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	<i>x</i>	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.*

## 0.5.2.3.35 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 <code>fprintf</code> . <i><code>colr_free()</code> will be called on any <a href="#">ColorArg</a>, <a href="#">ColorResult</a>, or <a href="#">ColorText</a> pointer passed to this function.</i>
----	-----	--

## Returns

Same as `fprintf`.

0.5.2.3.36 `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	<code>x</code>	A pointer to a supported type to free.
----	----------------	--

## Examples:

[ColorResult\\_example.c](#), [colr\\_join\\_example.c](#), and [colr\\_replace\\_all\\_example.c](#).

0.5.2.3.37 `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`

#### 0.5.2.3.38 `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	<code>x</code>	A supported type to build a string from.
----	----------------	--

#### 0.5.2.3.39 `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	<i>x</i>	A supported type to build a string from.
----	----------	--

#### 0.5.2.3.40 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	<i>x</i>	A supported type to build a string from.
----	----------	--

#### 0.5.2.3.41 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 <code>size_t</code> return value to check, from <code>mbrlen()</code> or <code>colr_mb_len()</code> .
----	---	---

## 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()`.

0.5.2.3.42 `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()`.

0.5.2.3.43 `colr_istr_eq`

```
#define colr_istr_eq(
    s1,
    s2 )
```

**Value:**

```
( \
    ((s1) && (s2)) ? \
        !strcasecmp((s1), (s2)) : \
        false \
)
```

Convenience macro for `!strcasecmp(s1, s2)`.

Parameters

in	<code>s1</code>	The first string to compare.
in	<code>s2</code>	The second string to compare.

Returns

1 if `s1` and `s2` are equal, otherwise 0.

#### 0.5.2.3.44 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	<code>joiner</code>	What to put between the other arguments. <code>ColorArg</code> pointer, <code>ColorResult</code> pointer, <code>ColorText</code> pointer, or string ( <code>char*</code> ).
in	...	Other arguments to join, with <code>joiner</code> between them. <code>ColorArg</code> pointers, <code>ColorResult</code> pointers, <code>ColorText</code> pointers, or strings, in any order. <i><code>colr_free()</code> will be called on any <code>ColorArg</code>, <code>ColorResult</code>, or <code>ColorText</code> pointer passed to this function.</i>

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](#).

#### 0.5.2.3.45 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	<i>joiner</i>	What to put between the other arguments. <a href="#">ColorArg</a> pointer, <a href="#">ColorText</a> pointer, or string.
in	...	Other arguments to join, with <i>joiner</i> between them. <a href="#">ColorArg</a> pointers, <a href="#">ColorText</a> 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](#).

#### 0.5.2.3.46 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	<i>x</i>	A supported type to build a string from.
----	----------	--

#### 0.5.2.3.47 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 it uses space as the default character.

Parameters

in	<i>text</i>	Text to colorize.
in	<i>justwidth</i>	Width for justification.
in	...	Fore, back, or style ColorArgs for <a href="#">Colr()</a> .

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](#).

## 0.5.2.3.48 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	<i>text</i>	Text to colorize.
in	<i>justwidth</i>	Width for justification.
in	<i>c</i>	The character to pad with.
in	...	Fore, back, or style ColorArgs for <a href="#">Colr()</a> .

## 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](#)

## 0.5.2.3.49 colr\_max

```
#define colr_max(
    a,
    b ) (a > b ? a : b)
```

Macro for (a > b ? a : b).

## Parameters

in	<i>a</i>	First value to compare.
in	<i>b</i>	Second value to compare.

Returns

a if a > b, otherwise b.

Referenced by `ColorText_length()`.

#### 0.5.2.3.50 `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 <code>colr_cat()</code> .
----	-----	---

#### 0.5.2.3.51 `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 <code>printf</code> . <i><code>colr_free()</code> will be called on any <code>ColorArg</code>, <code>ColorResult</code>, or <code>ColorText</code> pointer passed to this function.</i>
----	-----	--



## Returns

Same as `printf`.

## Examples:

[colr\\_printf\\_example.c](#).

0.5.2.3.52 `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	<i>func</i>	The standard <code>printf</code> function to call, with a return type of <code>int</code> .
in	...	Arguments for the <code>printf</code> function.

## Returns

Same as `func(...)`.

0.5.2.3.53 `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 <code>colr_cat()</code> .
----	-----	---

## Examples:

[colr\\_cat\\_example.c](#), [colr\\_join\\_example.c](#), and [simple\\_example.c](#).

0.5.2.3.54 `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	<i>s</i>	The string to operate on. <i>Must be null-terminated.</i>
in	<i>target</i>	A target string, regex pattern ( <code>regex_t</code> ), or regex match ( <code>regmatch_t</code> ) to replace in <i>s</i> . If a string is given, it <i>must be null-terminated</i> .
in	<i>repl</i>	A string, <a href="#">ColorArg</a> , <a href="#">ColorResult</a> , or <a href="#">ColorText</a> to replace the target string with. If this is NULL, then an empty string is used ("") as the replacement. <i><a href="#">colr_free()</a> will be called on any <a href="#">ColorArg</a>, <a href="#">ColorResult</a>, or <a href="#">ColorText</a> pointer passed to this function.</i>

## 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](#).

0.5.2.3.55 `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	<code>s</code>	The string to operate on. <i>Must be null-terminated.</i>
in	<code>target</code>	A target string, or regex pattern ( <code>regex_t</code> ) to replace in <code>s</code> . If a string is given, it <i>must be null-terminated.</i>
in	<code>repl</code>	A string, <code>ColorArg</code> , <code>ColorResult</code> , or <code>ColorText</code> to replace the target string with. If this is NULL, then an empty string is used ("" ) as the replacement. <i><code>colr_free()</code> will be called on any <code>ColorArg</code>, <code>ColorResult</code>, or <code>ColorText</code> pointer passed to this function.</i>

## 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](#).

## 0.5.2.3.56 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	<i>s</i>	The string to operate on. <i>Must be null-terminated.</i>
in	<i>target</i>	A regex pattern string (char*), regex pattern (regex_t), or regex match (regmatch_t) to replace in s. If a string is given, it <i>must be null-terminated</i> .
in	<i>repl</i>	A string, <a href="#">ColorArg</a> , <a href="#">ColorResult</a> , or <a href="#">ColorText</a> to replace the target string with. If this is NULL, then an empty string is used ("") as the replacement. <i>colr_free() will be called on any <a href="#">ColorArg</a>, <a href="#">ColorResult</a>, or <a href="#">ColorText</a> pointer passed to this function.</i>
in	<i>flags</i>	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](#).

## 0.5.2.3.57 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. <i>Must be null-terminated.</i>
in	target	A regex pattern string (char*), regex pattern (regex_t), or regex match (regmatch_t) to replace in s. If a string is given, it <i>must be null-terminated</i> .
in	repl	A string, <a href="#">ColorArg</a> , <a href="#">ColorResult</a> , or <a href="#">ColorText</a> to replace the target string with. If this is NULL, then an empty string is used ("") as the replacement. <i>colr_free() will be called on any <a href="#">ColorArg</a>, <a href="#">ColorResult</a>, or <a href="#">ColorText</a> pointer passed to this function.</i>
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](#)  
[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](#).

## 0.5.2.3.58 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	<i>x</i>	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()`.

## 0.5.2.3.59 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 it uses space as the default character.

## Parameters

in	<i>text</i>	Text to colorize.
in	<i>justwidth</i>	Width for justification.
in	...	Fore, back, or style ColorArgs for <a href="#">Colr()</a> .

## 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](#).

## 0.5.2.3.60 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	<i>text</i>	Text to colorize.
in	<i>justwidth</i>	Width for justification.
in	<i>c</i>	The character to pad with.
in	...	Fore, back, or style ColorArgs for <a href="#">Colr()</a> .

## 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](#)

## 0.5.2.3.61 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 <code>sprintf</code> . <i><code>colr_free()</code> will be called on any <a href="#">ColorArg</a>, <a href="#">ColorResult</a>, or <a href="#">ColorText</a> pointer passed to this function.</i>
----	-----	--

## Returns

Same as `sprintf`.

## Examples:

[colr\\_printf\\_example.c](#).

0.5.2.3.62 `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 <code>sprintf</code> . <i><code>colr_free()</code> will be called on any <a href="#">ColorArg</a>, <a href="#">ColorResult</a>, or <a href="#">ColorText</a> pointer passed to this function.</i>
----	-----	--

## Returns

Same as `sprintf`.

## Examples:

[colr\\_printf\\_example.c](#).

## 0.5.2.3.63 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	<i>s1</i>	The string to compare against the other two strings.
in	<i>s2</i>	The first string to compare with.
in	<i>s3</i>	The second string to compare with.

Returns

1 if *s1* is equal to *s2* or *s3*, otherwise 0.

## 0.5.2.3.64 colr\_str\_eq

```
#define colr_str_eq(
    s1,
    s2 )
```

**Value:**

```
( \
    ((s1) && (s2)) ? !strcmp((s1), (s2)) : false \
)
```

Convenience macro for `!strcmp(s1, s2)`.

Parameters

in	<i>s1</i>	The first string to compare.
in	<i>s2</i>	The second string to compare.

Returns

1 if *s1* and *s2* are equal, otherwise 0.

Referenced by `ColorResult_eq()`, and `RGB_from_str()`.

## 0.5.2.3.65 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](#).

## 0.5.2.3.66 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	<i>text</i>	String to colorize/style.
in	...	No more than 3 <a href="#">ColorArg</a> pointers for fore, back, and style in any order.

## Returns

An initialized [ColorText](#).

## See also

[Colr](#)

## 0.5.2.3.67 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	<i>s</i>	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](#).

## 0.5.2.3.68 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()`.

0.5.2.3.69 `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](#).

0.5.2.3.70 `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	<code>s</code>	A hex string to convert.
in	<code>default_value</code>	<code>ExtendedValue</code> 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`.

0.5.2.3.71 `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()`.



## 0.5.2.3.72 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\(\)](#).

## 0.5.2.3.73 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	<i>r</i>	The red value.
in	<i>g</i>	The green value.
in	<i>b</i>	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](#).

## 0.5.2.3.74 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	<i>rgbval</i>	The <a href="#">RGB</a> value to use.
----	---------------	---------------------------------------

## Returns

An [ExtendedValue](#) that closely matches the [RGB](#) value.

## See also

[ExtendedValue\\_from\\_RGB](#)  
[RGB\\_to\\_term\\_RGB](#)

0.5.2.3.75 `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	<i>x</i>	A <a href="#">BasicValue</a> , <a href="#">ExtendedValue</a> , or <a href="#">RGB</a> 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](#).

#### 0.5.2.3.76 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 its argument.

Uses `_Generic` (C11 standard) to dynamically create a [ColorArg](#). This is used by the [fore\(\)](#) macro.

Parameters

in	<code>x</code>	BasicValue, Extended (unsigned char), <a href="#">RGB</a> struct, or string (color name) for fore color.
----	----------------	--

Returns

A [ColorArg](#) with the FORE type set, and its `.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](#)

#### 0.5.2.3.77 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 <a href="#">RGB</a> struct.
----	---	---

## Returns

An allocated string.  
*You must `free()` the memory allocated by this function.*

## See also

[fore](#)  
[fore\\_arg](#)

0.5.2.3.78 `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 <a href="#">RGB</a> value.
----	---	--

## Returns

A stack-allocated escape code string.

## See also

[back\\_str\\_static](#)  
[style\\_str\\_static](#)  
[format\\_fg](#)  
[format\\_bg](#)

## 0.5.2.3.79 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](#).

## 0.5.2.3.80 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	<i>s</i>	A hex string to convert.
in	<i>default_rgb</i>	Default <a href="#">RGB</a> 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](#).

0.5.2.3.81 `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	<i>...</i>	Arguments for <code>asprintf</code> .
----	------------	---------------------------------------

0.5.2.3.82 `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	<i>r</i>	unsigned char Red value.
in	<i>g</i>	unsigned char Blue value.
in	<i>b</i>	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\(\)](#).

## 0.5.2.3.83 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	<i>x</i>	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](#).

#### 0.5.2.3.84 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	<i>x</i>	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](#)

#### 0.5.2.3.85 STYLE\_LEN\_MIN

```
#define STYLE_LEN_MIN 5
```

Minimum length for the shortest style escape code, including “\0”.

Use `STYLE_LEN` for allocation.

#### 0.5.2.3.86 style\_str

```
#define style_str(
    x ) ColorArg_to_esc(style_arg(x))
```

Return just the escape code string for a style.



## Parameters

in	<i>x</i>	StyleValue to use.
----	----------	--------------------

## Returns

An allocated string.  
*You must `free()` the memory allocated by this function.*

## See also

[style](#)  
[style\\_arg](#)

## 0.5.2.3.87 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	<i>x</i>	A StyleValue to use.
----	----------	----------------------

## Returns

A stack-allocated string.

See also

[fore\\_str\\_static](#)  
[back\\_str\\_static](#)  
[format\\_fg](#)  
[format\\_bg](#)

#### 0.5.2.3.88 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	<i>ap</i>	The <code>va_list</code> to use.
in	<i>vartype</i>	Expected type of the argument.
in	<i>x</i>	The variable to assign to (usually <code>arg</code> ).

Referenced by `_colr_join()`, `_colr_join_size()`, `ColorText_from_values()`, and `ColorText_set_values()`.

#### 0.5.2.4 Typedef Documentation

##### 0.5.2.4.1 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.

#### 0.5.2.5 Enumeration Type Documentation

##### 0.5.2.5.1 BasicValue

```
enum BasicValue
```

Basic color values, with a few convenience values for extended colors.

## 0.5.2.6 Function Documentation

0.5.2.6.1 `_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	<i>p</i>	Pointer to a heap-allocated object.
----	----------	-------------------------------------

0.5.2.6.2 `_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	<i>p</i>	The pointer to check.
----	----------	-----------------------

## Returns

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

0.5.2.6.3 `_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 `ColorArgs`, `ColorResults`, or `ColorTexts` 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	<code>joinerp</code>	The joiner (any <code>ColorArg</code> , <code>ColorResult</code> , <code>ColorText</code> , or string).
in	<code>...</code>	Zero or more <code>ColorArgs</code> , <code>ColorResults</code> , <code>ColorTexts</code> , 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.*

#### 0.5.2.6.4 \_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	<code>ps</code>	A NULL-terminated array of <code>ColorArgs</code> , <code>ColorResults</code> , <code>ColorTexts</code> , or strings ( <code>char*</code> ).
----	-----------------	--

#### Returns

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

Referenced by `colr_join_array()`.

#### 0.5.2.6.5 `_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	<i>joinerp</i>	The joiner (any <a href="#">ColorArg</a> , <a href="#">ColorResult</a> , <a href="#">ColorText</a> , or string).
in	<i>ps</i>	An array of pointers to <a href="#">ColorArgs</a> , <a href="#">ColorResults</a> , <a href="#">ColorTexts</a> , or strings. The array must have NULL as the last item if count is greater than the total number of items.
in	<i>count</i>	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()`.

#### 0.5.2.6.6 `_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	<i>joinerp</i>	The joiner (any <a href="#">ColorArg</a> , <a href="#">ColorText</a> , or string (char*)).
in	<i>args</i>	A <i>va_list</i> with zero or more <a href="#">ColorArgs</a> , <a href="#">ColorTexts</a> , 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()`.

0.5.2.6.7 `_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	<i>p</i>	A <a href="#">ColorArg</a> pointer, <a href="#">ColorResult</a> pointer, <a href="#">ColorText</a> 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()`.

0.5.2.6.8 `_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	<i>p</i>	A <a href="#">ColorArg</a> pointer, <a href="#">ColorResult</a> pointer, <a href="#">ColorText</a> 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](#)

#### 0.5.2.6.9 `_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	<i>p</i>	A <a href="#">ColorArg</a> pointer, <a href="#">ColorResult</a> pointer, <a href="#">ColorText</a> 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.*

0.5.2.6.10 `_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	<i>fmter</i>	A formatter function ( <code>RGB_fmter</code> ) that can create escape codes from <code>RGB</code> values.
in	<i>s</i>	The string to "rainbowize". Input <i>must be null-terminated</i> .
in	<i>freq</i>	The "tightness" for colors.
in	<i>offset</i>	The starting offset into the rainbow.
in	<i>spread</i>	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()`.

0.5.2.6.11 `ArgType_eq()`

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

Compares two `ArgTypes`.

This is used to implement `colr_eq()`.

## Parameters

in	<i>a</i>	The first <code>ArgType</code> to compare.
in	<i>b</i>	The second <code>ArgType</code> to compare.



## Returns

true if they are equal, otherwise false.

## 0.5.2.6.12 ArgType\_repr()

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

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

## Parameters

in	<i>type</i>	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().

## 0.5.2.6.13 ArgType\_to\_str()

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

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

## Parameters

in	<i>type</i>	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()`.

#### 0.5.2.6.14 BasicValue\_eq()

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

Compares two BasicValues.

This is used to implement `colr_eq()`.

Parameters

in	<i>a</i>	The first BasicValue to compare.
in	<i>b</i>	The second BasicValue to compare.

Returns

true if they are equal, otherwise false.

See also

[BasicValue](#)

#### 0.5.2.6.15 BasicValue\_from\_esc()

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

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

Parameters

in	<i>s</i>	Escape-code string. <i>Must be null-terminated.</i>
----	----------	--

Return values

<i>BasicValue</i>	value on success.
<i>BASIC_INVALID</i>	on error (or if <i>s</i> is NULL).
<i>BASIC_INVALID_RANGE</i>	if the code number was outside of the range 0–255.

See also

[BasicValue](#)

#### 0.5.2.6.16 BasicValue\_from\_str()

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

Convert named argument to an actual BasicValue enum value.

Parameters

in	<i>arg</i>	Color name to find the BasicValue for.
----	------------	--

Returns

BasicValue value on success, or BASIC\_INVALID on error.

See also

[BasicValue](#)

#### 0.5.2.6.17 BasicValue\_is\_invalid()

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

Determines whether a BasicValue is invalid.

Parameters

in	<i>bval</i>	A BasicValue to check.
----	-------------	------------------------

Returns

true if the value is considered invalid, otherwise false.

See also

[BasicValue](#)

Referenced by ExtendedValue\_from\_BasicValue().

## 0.5.2.6.18 BasicValue\_is\_valid()

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

Determines whether a BasicValue is valid.

Parameters

in	<i>bval</i>	A BasicValue to check.
----	-------------	------------------------

Returns

true if the value is considered valid, otherwise false.

See also

[BasicValue](#)

## 0.5.2.6.19 BasicValue\_repr()

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

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

Parameters

in	<i>bval</i>	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](#)

## 0.5.2.6.20 BasicValue\_to\_ansi()

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

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

## Parameters

in	<i>type</i>	ArgType (FORE/BACK).
in	<i>bval</i>	BasicValue to convert.

## Returns

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

## See also

[BasicValue](#)

Referenced by `format_bg()`, and `format_fg()`.

## 0.5.2.6.21 BasicValue\_to\_str()

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

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

## Parameters

in	<i>bval</i>	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](#)

## 0.5.2.6.22 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](#)

#### 0.5.2.6.23 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	<i>a</i>	First <a href="#">ColorArg</a> to compare.
in	<i>b</i>	Second <a href="#">ColorArg</a> to compare.

Returns

true if they are equal, otherwise false.

See also

[ColorArg](#)

Referenced by `ColorText_has_arg()`.

#### 0.5.2.6.24 ColorArg\_example()

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

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

Parameters

in	<i>carg</i>	A <a href="#">ColorArg</a> to get an example string for.
in	<i>colored</i>	Whether to include a colored example. If set to false, there will be no escape-codes in the string.
Generated by Doxygen		

## 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](#)

0.5.2.6.25 `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 [color\\_free\(\)](#) macro).

## Parameters

in	<i>p</i>	<a href="#">ColorArg</a> 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()`.

0.5.2.6.26 `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	<i>type</i>	<code>ArgType</code> (FORE, BACK, STYLE).
in	<i>value</i>	<code>BasicValue</code> to use.

## Returns

A [ColorArg](#), with the `.value.type` member possibly set to `TYPE_INVALID`.

## See also

[ColorArg](#)

0.5.2.6.27 `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	<code>s</code>	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()`.

0.5.2.6.28 `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	<i>type</i>	ArgType (FORE, BACK, STYLE).
in	<i>value</i>	ExtendedValue to use.

## Returns

A [ColorArg](#), with the `.value.type` member possibly set to `TYPE_INVALID`.

## See also

[ColorArg](#)

## 0.5.2.6.29 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	<i>type</i>	ArgType (FORE, BACK, STYLE).
in	<i>value</i>	<a href="#">RGB</a> struct to use.

## Returns

A [ColorArg](#), with the `.value.type` member possibly set to `TYPE_INVALID`.

## See also

[ColorArg](#)

## 0.5.2.6.30 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	<i>type</i>	ArgType (FORE, BACK, STYLE).
in	<i>colorname</i>	A known color name/style.

## Returns

A [ColorArg](#) struct with usable values.

## See also

[ColorArg](#)

## 0.5.2.6.31 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	<i>type</i>	ArgType (FORE, BACK, STYLE).
in	<i>value</i>	StyleValue to use.

## Returns

A [ColorArg](#), with the `.value.type` member possibly set to TYPE\_INVALID.

## See also

[ColorArg](#)

## 0.5.2.6.32 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	<i>type</i>	ArgType value, to mark the type of <a href="#">ColorArg</a> .
in	<i>colrtype</i>	ColorType value, to mark the type of <a href="#">ColorValue</a> .
in	<i>p</i>	A pointer to either a BasicValue, ExtendedValue, or a <a href="#">RGB</a> .

## 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](#)

## 0.5.2.6.33 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	<i>carg</i>	A <a href="#">ColorArg</a> 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()`.

## 0.5.2.6.34 ColorArg\_is\_invalid()

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

Checks to see if a [ColorArg](#) holds an invalid value.

## Parameters

in	<i>carg</i>	<a href="#">ColorArg</a> struct to check.
----	-------------	---

## Returns

true if the value is invalid, otherwise false.

## See also

[ColorArg](#)

0.5.2.6.35 [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	<i>p</i>	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\(\)](#).

0.5.2.6.36 [ColorArg\\_is\\_valid\(\)](#)

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

Checks to see if a [ColorArg](#) holds a valid value.

## Parameters

in	<i>carg</i>	<a href="#">ColorArg</a> struct to check.
----	-------------	---

## Returns

true if the value is valid, otherwise false.

## See also

[ColorArg](#)

## 0.5.2.6.37 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	<i>carg</i>	<a href="#">ColorArg</a> 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\(\)](#).

## 0.5.2.6.38 ColorArg\_repr()

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

Creates a string (char\*) representation for a [ColorArg](#).

Allocates memory for the string representation.

## Parameters

in	carg	<a href="#">ColorArg</a> 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()`.

0.5.2.6.39 `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	<a href="#">ColorArg</a> to get the ArgType and <a href="#">ColorValue</a> 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()`.

## 0.5.2.6.40 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. <i>Must have room for the code type being used.</i> See <a href="#">ColorArg_length()</a> for determining the size needed.
in	carg	<a href="#">ColorArg</a> to get the ArgType and <a href="#">ColorValue</a> from.

## Returns

true if the [ColorArg](#) was valid, otherwise false.

## See also

[ColorArg](#)

## 0.5.2.6.41 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	<a href="#">ColorArg</a> 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()`.

#### 0.5.2.6.42 `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	<i>ps</i>	A pointer to an array of <code>ColorArgs</code> , where NULL is the last item.
----	-----------	--

#### 0.5.2.6.43 `ColorArgs_array_repr()`

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

Creates a string representation for an array of [ColorArg](#) pointers.

Parameters

in	<i>lst</i>	The <a href="#">ColorArg</a> array to create the representation for ( <code>ColorArg**</code> ).
----	------------	--

Returns

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

#### 0.5.2.6.44 `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	<code>s</code>	A string to get the escape-codes from. <i>Must be null-terminated.</i>
in	<i>unique</i>	Whether to only include <i>unique</i> 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

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

## 0.5.2.6.45 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().

## 0.5.2.6.46 ColorJustify\_eq()

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

Compares two [ColorJustify](#) structs.

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

## Parameters

in	<i>a</i>	First <a href="#">ColorJustify</a> to compare.
in	<i>b</i>	Second <a href="#">ColorJustify</a> to compare.

## Returns

true if they are equal, otherwise false.

## See also

[ColorJustify](#)

0.5.2.6.47 [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	<i>cjust</i>	The <a href="#">ColorJustify</a> to check.
----	--------------	--

## Returns

true if the [ColorJustify](#) is empty, otherwise false.

## See also

[ColorJustify](#)  
[ColorJustify\\_empty](#)

Referenced by [ColorText\\_is\\_empty\(\)](#), and [ColorText\\_length\(\)](#).

0.5.2.6.48 [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	<i>method</i>	ColorJustifyMethod to use.
in	<i>width</i>	Width for justification. If 0 is given, <a href="#">ColorText</a> will use the width from <a href="#">colr_term_size()</a> .
in	<i>padchar</i>	Padding character to use. If 0 is given, the default, space (" "), is used.

## Returns

An initialized [ColorJustify](#).

## 0.5.2.6.49 ColorJustify\_repr()

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

Creates a string (char\*) representation for a [ColorJustify](#).

Allocates memory for the string representation.

## Parameters

in	<i>cjust</i>	<a href="#">ColorJustify</a> 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\(\)](#).

## 0.5.2.6.50 ColorJustifyMethod\_repr()

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

Creates a string (char\*) representation for a [ColorJustifyMethod](#).

Allocates memory for the string representation.

## Parameters

in	<i>meth</i>	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().

## 0.5.2.6.51 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().

## 0.5.2.6.52 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	<i>a</i>	First <a href="#">ColorResult</a> to compare.
in	<i>b</i>	Second <a href="#">ColorResult</a> to compare.

## Returns

true if they are equal, otherwise false.

## See also

[ColorResult](#)

## 0.5.2.6.53 ColorResult\_free()

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

Free allocated memory for a [ColorResult](#) and it's .result member.

## Parameters

in	<i>p</i>	A <a href="#">ColorResult</a> 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().

## 0.5.2.6.54 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	<i>p</i>	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()`.

#### 0.5.2.6.55 `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	<i>cres</i>	A <a href="#">ColorResult</a> 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()`.

#### 0.5.2.6.56 `ColorResult_new()`

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

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

Parameters

in	<i>s</i>	An allocated string to use for the <code>.result</code> member.
----	----------	---

Returns

An initialized [ColorResult](#).

See also

[ColorResult](#)

#### 0.5.2.6.57 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	<i>cres</i>	A <a href="#">ColorResult</a> 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()`.

#### 0.5.2.6.58 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	<i>cres</i>	A <a href="#">ColorResult</a> 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](#)

#### 0.5.2.6.59 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	<i>cres</i>	A <a href="#">ColorResult</a> to use.
----	-------------	---------------------------------------

Returns

A stringified-version of 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\(\)](#).

#### 0.5.2.6.60 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\(\)](#).



## 0.5.2.6.61 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	<i>p</i>	Pointer to <a href="#">ColorText</a> 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\(\)](#).

## 0.5.2.6.62 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	<i>p</i>	Pointer to a <a href="#">ColorText</a> .
----	----------	--

See also

[ColorText](#)

Referenced by [ColorText\\_free\(\)](#).

## 0.5.2.6.63 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	<i>text</i>	Text to colorize (a regular string).
in	...	<a href="#">ColorArgs</a> for fore, back, and style, in any order.

Returns

An initialized [ColorText](#) struct.

See also

[ColorText](#)

#### 0.5.2.6.64 [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	<i>ctext</i>	The <a href="#">ColorText</a> to inspect.
in	<i>carg</i>	The <a href="#">ColorArg</a> to look for.

Returns

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

See also

[ColorText](#)

#### 0.5.2.6.65 [ColorText\\_has\\_args\(\)](#)

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

Checks to see if a [ColorText](#) has any argument values set.

## Parameters

in	<i>ctext</i>	A <a href="#">ColorText</a> to check.
----	--------------	---------------------------------------

## Returns

true if `.fore`, `.back`, or `.style` is set to a non-empty [ColorArg](#), otherwise false.

## See also

[ColorText](#)

0.5.2.6.66 `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	<i>ctext</i>	The <a href="#">ColorText</a> to check.
----	--------------	---

## Returns

true if the [ColorText](#) is empty, otherwise false.

## See also

[ColorText](#)  
[ColorText\\_empty](#)

0.5.2.6.67 `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	<i>p</i>	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()`.

0.5.2.6.68 `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	<i>ctext</i>	<a href="#">ColorText</a> 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()`.

0.5.2.6.69 `ColorText_repr()`

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

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

## Parameters

in	<i>ctext</i>	<a href="#">ColorText</a> to get the string representation for.
----	--------------	---

## Returns

Allocated string for the [ColorText](#).

## See also

[ColorText](#)

Referenced by `_colr_ptr_repr()`.

0.5.2.6.70 `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	<i>ctext</i>	The <a href="#">ColorText</a> to set the justification method for.
in	<i>cjust</i>	The <a href="#">ColorJustify</a> struct to use.

## Returns

The same pointer that was given as *ctext*.

## See also

[ColorText](#)

0.5.2.6.71 `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	<i>ctext</i>	A <a href="#">ColorText</a> to initialize with values.
in	<i>text</i>	Text to colorize (a regular string).
in	...	A <code>va_list</code> with <code>ColorArgs</code> pointers for fore, back, and style, in any order.

## Returns

An initialized [ColorText](#) struct.

## See also

[ColorText](#)

0.5.2.6.72 `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	<i>ctext</i>	<a href="#">ColorText</a> 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](#)

0.5.2.6.73 `ColorText_to_str()`

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

Stringifies a [ColorText](#) struct, creating a mix of escape codes and text.

## Parameters

in	<i>c</i> text	<a href="#">ColorText</a> 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()`.

0.5.2.6.74 `ColorType_eq()`

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

Compares two `ColorTypes`.

This is used to implement [colr\\_eq\(\)](#).

## Parameters

in	<i>a</i>	The first <code>ColorType</code> to compare.
in	<i>b</i>	The second <code>ColorType</code> to compare.

## Returns

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

## See also

[ColorType](#)

## 0.5.2.6.75 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	<i>arg</i>	Color name to get the ColorType for.
----	------------	--------------------------------------

Return values

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

See also

[ColorType](#)

## 0.5.2.6.76 ColorType\_is\_invalid()

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

Check to see if a ColorType value is considered invalid.

Parameters

in	<i>type</i>	ColorType value to check.
----	-------------	---------------------------

Returns

true if the value is considered invalid, otherwise false.



See also

[ColorType](#)

#### 0.5.2.6.77 ColorType\_is\_valid()

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

Check to see if a ColorType value is considered valid.

Parameters

in	<i>type</i>	ColorType value to check.
----	-------------	---------------------------

Returns

true if the value is considered valid, otherwise false.

See also

[ColorType](#)

#### 0.5.2.6.78 ColorType\_repr()

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

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

Parameters

in	<i>type</i>	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](#)

## 0.5.2.6.79 ColorType\_to\_str()

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

Create a human-friendly string (char\*) representation for a [ColorType](#).

Parameters

in	<i>type</i>	A <a href="#">ColorType</a> 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\(\)](#).

## 0.5.2.6.80 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](#)

## 0.5.2.6.81 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	<i>a</i>	First <a href="#">ColorValue</a> to compare.
in	<i>b</i>	Second <a href="#">ColorValue</a> to compare.

## Returns

true if they are equal, otherwise false.

## See also

[ColorValue](#)

Referenced by `ColorArg_eq()`.

0.5.2.6.82 `ColorValue_example()`

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

Create a string (char\*) representation of a [ColorValue](#) with a human-friendly type/name.

## Parameters

in	<i>cval</i>	A <a href="#">ColorValue</a> 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()`.

0.5.2.6.83 `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. <i>Must be null-terminated.</i>
----	---	--

## 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: <ul style="list-style-type: none"><li>• TYPE_INVALID</li><li>• TYPE_INVALID_EXT_RANGE</li><li>• TYPE_INVALID_RGB_RANGE</li></ul>
-----	---

## See also

[ColorValue](#)  
[ColorArg\\_from\\_esc](#)

Referenced by [ColorArg\\_from\\_esc\(\)](#).

0.5.2.6.84 [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 <a href="#">RGB</a> string).
----	---	---

## Returns

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

Return values

<i>For</i>	invalid strings, the <code>.type</code> member can be one of: <ul style="list-style-type: none"> <li>• <code>TYPE_INVALID</code></li> <li>• <code>TYPE_INVALID_EXT_RANGE</code></li> <li>• <code>TYPE_INVALID_RGB_RANGE</code></li> </ul>
------------	---

See also

[ColorValue](#)

Referenced by `ColorArg_from_str()`.

#### 0.5.2.6.85 `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	<i>type</i>	A <code>ColorType</code> value, to mark the type of <a href="#">ColorValue</a> .
in	<i>p</i>	A pointer to either a <code>BasicValue</code> , <code>ExtendedValue</code> , or a <a href="#">RGB</a> .

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()`.

#### 0.5.2.6.86 `ColorValue_has_BasicValue()`

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

Checks to see if a [ColorValue](#) has a `BasicValue` set.

## Parameters

in	<i>cval</i>	<a href="#">ColorValue</a> to check.
in	<i>bval</i>	BasicValue to look for.

## Returns

true if the [ColorValue](#) has the exact BasicValue set.

## See also

[ColorValue](#)

0.5.2.6.87 [ColorValue\\_has\\_ExtendedValue\(\)](#)

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

Checks to see if a [ColorValue](#) has a ExtendedValue set.

## Parameters

in	<i>cval</i>	<a href="#">ColorValue</a> to check.
in	<i>eval</i>	ExtendedValue to look for.

## Returns

true if the [ColorValue](#) has the exact ExtendedValue set.

## See also

[ColorValue](#)

0.5.2.6.88 [ColorValue\\_has\\_RGB\(\)](#)

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

Checks to see if a [ColorValue](#) has a [RGB](#) value set.

## Parameters

in	<i>cval</i>	<a href="#">ColorValue</a> to check.
in	<i>rgb</i>	<a href="#">RGB</a> value to look for.

## Returns

true if the [ColorValue](#) has the exact [RGB](#) value set.

## See also

[ColorValue](#)

0.5.2.6.89 [ColorValue\\_has\\_StyleValue\(\)](#)

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

Checks to see if a [ColorValue](#) has a [StyleValue](#) set.

## Parameters

in	<i>cval</i>	<a href="#">ColorValue</a> to check.
in	<i>sval</i>	<a href="#">StyleValue</a> to look for.

## Returns

true if the [ColorValue](#) has the exact [StyleValue](#) set.

## See also

[ColorValue](#)

0.5.2.6.90 [ColorValue\\_is\\_empty\(\)](#)

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

Checks to see if a [ColorValue](#) is an empty placeholder.

## Parameters

in	<i>cval</i>	<a href="#">ColorValue</a> to check.
----	-------------	--------------------------------------

## Returns

true if the [ColorValue](#) is "empty", otherwise false.

See also

[ColorValue](#)  
[ColorValue\\_empty](#)  
[ColorArg\\_empty](#)  
[ColorArg\\_is\\_empty](#)

#### 0.5.2.6.91 ColorValue\_is\_invalid()

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

Checks to see if a [ColorValue](#) holds an invalid value.

Parameters

in	<i>cval</i>	<a href="#">ColorValue</a> struct to check.
----	-------------	---

Returns

true if the value is invalid, otherwise false.

See also

[ColorValue](#)

Referenced by [ColorArg\\_from\\_esc\(\)](#).

#### 0.5.2.6.92 ColorValue\_is\_valid()

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

Checks to see if a [ColorValue](#) holds a valid value.

Parameters

in	<i>cval</i>	<a href="#">ColorValue</a> struct to check.
----	-------------	---

Returns

true if the value is valid, otherwise false.

See also

[ColorValue](#)



## 0.5.2.6.93 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	<i>type</i>	ArgType (FORE, BACK, STYLE)
in	<i>cval</i>	<a href="#">ColorValue</a> 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().

## 0.5.2.6.94 ColorValue\_repr()

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

Creates a string (char\*) representation of a [ColorValue](#).

Parameters

in	<i>cval</i>	A <a href="#">ColorValue</a> 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().

## 0.5.2.6.95 ColorValue\_to\_esc()

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

Converts a [ColorValue](#) into an escape code string (char\*).

## Parameters

in	<i>type</i>	ArgType (FORE, BACK, STYLE) to build the escape code for.
in	<i>cval</i>	<a href="#">ColorValue</a> 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().

## 0.5.2.6.96 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	<i>dest</i>	Destination string for the escape code string. <i>Must have room for the code type being used.</i>
in	<i>type</i>	ArgType (FORE, BACK, STYLE) to build the escape code for.
in	<i>cval</i>	<a href="#">ColorValue</a> 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()`.

#### 0.5.2.6.97 `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	<i>match</i>	A <code>regmatch_t</code> to allocate for and copy.
----	--------------	---

Returns

An allocated copy of the `regmatch_t`.

Referenced by `colr_re_matches()`.

#### 0.5.2.6.98 `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	<i>s</i>	The string to append to. <i>Must have extra room for <code>CODE_RESET_ALL</code>. Must be null-terminated.</i>
----	----------	--

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

#### 0.5.2.6.99 `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
\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()`.

#### 0.5.2.6.100 `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. Input <i>must be null-terminated</i> .

#### Returns

true if c is found in s, otherwise false.

Referenced by `colr_str_chars_lcount()`, and `colr_str_lstrip_chars()`.

0.5.2.6.101 `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](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()`.

0.5.2.6.102 `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()`.

0.5.2.6.103 `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
<code>\'</code>	single quote
<code>\"</code>	double quote
<code>\?</code>	question mark
<code>\\</code>	backslash
<code>\a</code>	audible bell
<code>\b</code>	backspace
<code>\f</code>	form feed - new page
<code>\n</code>	line feed - new line
<code>\r</code>	carriage return
<code>\t</code>	horizontal tab
<code>\v</code>	vertical tab

## Parameters

in	<code>c</code>	The character to check.
----	----------------	-------------------------

## Returns

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

Referenced by `colr_str_repr()`.

0.5.2.6.104 `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	<i>marker</i>	A colr marker, like <code>COLORARG_MARKER</code> , <code>COLORTTEXT_MARKER</code> , etc.
in	<i>p</i>	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()`.

0.5.2.6.105 `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()`.

0.5.2.6.106 `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	<i>matches</i>	A pointer to an array of <code>regmatch_t</code> pointers.
-----	----------------	--

Referenced by `colr_str_replace_re_pat_all()`.

0.5.2.6.107 `colr_join_array()`

```
char* colr_join_array (
```

```
void * joinerp,
void * ps )
```

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

Parameters

in	<i>joinerp</i>	The joiner (any <a href="#">ColorArg</a> , <a href="#">ColorText</a> , or string (char*)).
in	<i>ps</i>	An array of pointers to <a href="#">ColorArgs</a> , <a href="#">ColorTexts</a> , 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](#)

#### 0.5.2.6.108 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	<i>joinerp</i>	The joiner (any <a href="#">ColorArg</a> , <a href="#">ColorText</a> , or string (char*)).
in	<i>ps</i>	An array of pointers to <a href="#">ColorArgs</a> , <a href="#">ColorTexts</a> , or strings (char*). The array must have at least a length of count, unless a NULL element is placed at the end.
in	<i>count</i>	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\(\)](#).

#### 0.5.2.6.109 [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	<i>s</i>	The string to check.
in	<i>length</i>	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 <i>s</i> is NULL/empty, or <i>length</i> is 0.
( <i>size_t</i> )-1	if an invalid multibyte sequence is found at the start of <i>s</i> .

See also

[colr\\_str\\_mb\\_len](#)  
[colr\\_is\\_valid\\_mblen](#)

Referenced by [\\_rainbow\(\)](#).

#### 0.5.2.6.110 [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	<i>fp</i>	FILE pointer for output.
in	<i>info</i>	Info from printf about how to format the argument.
in	<i>args</i>	Argument list (with only 1 argument), containing a <a href="#">ColorArg</a> , <a href="#">ColorResult</a> , <a href="#">ColorText</a> , or string (char*) to format.

## Returns

The number of characters written.

Referenced by `colr_printf_register()`.

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

## Returns

The number of argument types set in `argtypes`.

Referenced by `colr_printf_register()`.

#### 0.5.2.6.112 `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`.

#### 0.5.2.6.113 `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	<code>s</code>	The string to search.
in	<code>repattern</code>	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()`.

#### 0.5.2.6.114 `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()`.

0.5.2.6.115 `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	<i>lst</i>	The string array to look in.
in	<i>s</i>	The string to look for.

## Returns

true if the string is found, otherwise false.

## Return values

<code>&lt;tt&gt;false&lt;/tt&gt;</code>	if <i>lst</i> is <code>NULL</code> or <i>s</i> is <code>NULL</code> .
---	---

Referenced by `colr_str_get_codes()`.

0.5.2.6.116 `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	<i>ps</i>	A pointer to an array of strings.
----	-----------	-----------------------------------

Referenced by `ColorArgs_from_str()`.

## 0.5.2.6.117 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	<i>s</i>	The string to justify. Input <i>must be null-terminated</i> .
in	<i>width</i>	The overall width for the resulting string. If set to '0', the terminal width will be used from <a href="#">colr_term_size()</a> .
in	<i>padchar</i>	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\(\)](#).

## 0.5.2.6.118 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	<i>s</i>	The string to examine. <i>Must be null-terminated.</i>
in	<i>c</i>	The character to count. <i>Must not be 0.</i>

## Returns

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

Referenced by `_rainbow()`.

0.5.2.6.119 `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	<code>s</code>	The string to examine. <i>Must be null-terminated.</i>
in	<code>c</code>	The character to count. <i>Must not be 0.</i>

## Returns

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

Referenced by `colr_str_lstrip_char()`.

0.5.2.6.120 `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	<code>s</code>	The string to examine. <i>Must be null-terminated.</i>
in	<code>chars</code>	The characters to count, in any order. <i>Must not be 0.</i>

## Returns

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

Referenced by `colr_str_lstrip_chars()`.

0.5.2.6.121 `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	<code>s</code>	A string to count the escape-codes for. <i>Must be null-terminated.</i>
----	----------------	--

## 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()`.

0.5.2.6.122 `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	<code>s</code>	A string to count the code-chars for. <i>Must be null-terminated.</i>
----	----------------	--

## Returns

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

0.5.2.6.123 `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. <i>Must have room for strlen(src) + 1 or length + 1.</i>
in	src	Source string to copy.
in	length	Maximum characters to copy. <i>This does not include the null-terminator.</i> Usually set to strlen(dest).

Returns

On success, a pointer to dest is returned.

#### 0.5.2.6.124 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. <i>Must be null-terminated.</i>
in	suffix	Suffix to check for. <i>Must be null-terminated.</i>

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().



## 0.5.2.6.125 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	<code>s</code>	A string to get the escape-codes from. <i>Must be null-terminated.</i>
in	<code>unique</code>	Whether to only include <i>unique</i> 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

<i>If</i>	<code>s</code> 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.
<i>On</i>	success, there will be at least two pointers behind the return value. The last pointer is always NULL.

Referenced by `ColorArgs_from_str()`.

## 0.5.2.6.126 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	<code>s</code>	The string to check. Can be NULL. Input <i>must be null-terminated.</i>
----	----------------	--

## Returns

true if the string has at least one escape code, otherwise false.

## See also

[colr\\_str\\_is\\_codes](#)

## 0.5.2.6.127 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. <i>Must be null-terminated.</i>
----	---	--

## 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()`.

## 0.5.2.6.128 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.

0.5.2.6.129 `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. Input <i>must be null-terminated</i> .
----	---	--

## Returns

true if the string is escape-codes only, otherwise false.

See also

[colr\\_str\\_has\\_codes](#)

0.5.2.6.130 `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. Input <i>must be null-terminated</i> .
----	---	--

## Returns

true if all characters are digits (0-9), otherwise false.

Referenced by `ExtendedValue_from_str()`.

0.5.2.6.131 `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	<code>s</code>	The string to justify. Input <i>must be null-terminated</i> .
in	<code>width</code>	The overall width for the resulting string. If set to '0', the terminal width will be used from <a href="#">colr_term_size()</a> .
in	<code>padchar</code>	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()`.

0.5.2.6.132 `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	<code>s</code>	The input string to convert to lower case.
----	----------------	--

## 0.5.2.6.133 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	<i>dest</i>	Destination char array. Must have room for <code>strlen(s) + 1</code> .
in	<i>s</i>	String to strip the character from.
in	<i>length</i>	Length of <i>s</i> , the input string.
in	<i>c</i>	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()`.

## 0.5.2.6.134 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	<i>s</i>	String to strip the character from.
in	<i>c</i>	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.*

0.5.2.6.135 `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	<code>s</code>	The string to strip. <i>s</i> <i>Must be null-terminated</i> .
in	<code>chars</code>	A string of characters to remove. Each will be removed from the start of the string. <i>chars</i> <i>Must be null-terminated</i> .

## 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.*

0.5.2.6.136 `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	<code>s</code>	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()`.

## 0.5.2.6.137 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	<i>s</i>	String to get the length for. Input <i>must be null-terminated</i> .
----	----------	---

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()`.

## 0.5.2.6.138 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	<i>s</i>	The string to operate on.
in	<i>target</i>	The string to replace.
in	<i>repl</i>	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()`.

#### 0.5.2.6.139 `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	<code>s</code>	The string to operate on.
in	<code>target</code>	The string to replace.
in	<code>repl</code>	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()`.

#### 0.5.2.6.140 `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	<i>s</i>	The string to operate on.
in	<i>target</i>	The string to replace.
in	<i>repl</i>	The <a href="#">ColorArg</a> to produce escape-codes to replace with. <a href="#">ColorArg_free()</a> 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](#)

## 0.5.2.6.141 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	<i>s</i>	The string to operate on.
in	<i>target</i>	The string to replace.
in	<i>repl</i>	The <a href="#">ColorResult</a> to produce escape-codes to replace with. <a href="#">ColorResult_free()</a> 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](#)

0.5.2.6.142 `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	<code>s</code>	The string to operate on.
in	<code>target</code>	The string to replace.
in	<code>repl</code>	The <a href="#">ColorText</a> to produce text/escape-codes to replace with. <a href="#">ColorText_free()</a> 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](#)

0.5.2.6.143 `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	<code>s</code>	The string to operate on.
in	<code>target</code>	The string to replace.
in	<code>repl</code>	The string to replace with.
in	<code>count</code>	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()`.

0.5.2.6.144 `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	<code>s</code>	The string to operate on.
in	<code>target</code>	The string to replace.
in	<code>repl</code>	The <a href="#">ColorArg</a> to produce escape-codes to replace with. <a href="#">ColorArg_free()</a> 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](#)

0.5.2.6.145 `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	<i>s</i>	The string to operate on.
in	<i>target</i>	The string to replace.
in	<i>repl</i>	The <a href="#">ColorResult</a> to produce escape-codes to replace with. <a href="#">ColorResult_free()</a> 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](#)

## 0.5.2.6.146 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	<i>s</i>	The string to operate on.
in	<i>target</i>	The string to replace.
in	<i>repl</i>	The <a href="#">ColorText</a> to produce text/escape-codes to replace with. <a href="#">ColorText_free()</a> 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](#)

0.5.2.6.147 `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	<code>s</code>	The string to operate on.
in	<code>pattern</code>	The regex match object to find text to replace.
in	<code>repl</code>	The string to replace with.
in	<code>re_flags</code>	Flags for <code>regcomp()</code> . <code>REG_EXTENDED</code> 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()`.

0.5.2.6.148 `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	<i>s</i>	The string to operate on.
in	<i>pattern</i>	The regex match object to find text to replace.
in	<i>repl</i>	The string to replace with.
in	<i>re_flags</i>	Flags for <code>regcomp()</code> . <code>REG_EXTENDED</code> 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()`.

0.5.2.6.149 `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	<i>s</i>	The string to operate on.
in	<i>pattern</i>	The regex pattern to compile.
in	<i>repl</i>	The <a href="#">ColorArg</a> to produce escape-codes to replace with. <a href="#">ColorArg_free()</a> is called after the replacement is done.
in	<i>re_flags</i>	Flags for <code>regcomp()</code> . <code>REG_EXTENDED</code> 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](#)

## 0.5.2.6.150 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	<i>s</i>	The string to operate on.
in	<i>pattern</i>	The regex match object to find text to replace.
in	<i>repl</i>	The <a href="#">ColorResult</a> to produce escape-codes to replace with. <a href="#">ColorResult_free()</a> is called after the replacement is done.
in	<i>re_flags</i>	Flags for <code>regcomp()</code> . 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](#)

0.5.2.6.151 `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	<code>s</code>	The string to operate on.
in	<code>pattern</code>	The regex match object to find text to replace.
in	<code>repl</code>	The <code>ColorText</code> to produce text/escape-codes to replace with. <code>ColorText_free()</code> is called after the replacement is done.
in	<code>re_flags</code>	Flags for <code>regcomp()</code> . <code>REG_EXTENDED</code> 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`

0.5.2.6.152 `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	<code>s</code>	The string to operate on.
in	<code>pattern</code>	The regex pattern to compile.
in	<code>repl</code>	The <code>ColorArg</code> to produce escape-codes to replace with. <code>ColorArg_free()</code> is called after the replacement is done.
in	<code>re_flags</code>	Flags for <code>regcomp()</code> . <code>REG_EXTENDED</code> 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](#)

## 0.5.2.6.153 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	<i>s</i>	The string to operate on.
in	<i>pattern</i>	The regex match object to find text to replace.
in	<i>repl</i>	The <a href="#">ColorResult</a> to produce escape-codes to replace with. <a href="#">ColorResult_free()</a> is called after the replacement is done.
in	<i>re_flags</i>	Flags for <code>regcomp()</code> . 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](#)

0.5.2.6.154 `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	<code>s</code>	The string to operate on.
in	<code>pattern</code>	The regex match object to find text to replace.
in	<code>repl</code>	The <code>ColorText</code> to produce text/escape-codes to replace with. <code>ColorText_free()</code> is called after the replacement is done.
in	<code>re_flags</code>	Flags for <code>regcomp()</code> . <code>REG_EXTENDED</code> 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](#)

0.5.2.6.155 `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	<code>s</code>	The string to operate on.
in	<code>match</code>	The regex match object to find text to replace.
in	<code>repl</code>	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\(\)](#).

## 0.5.2.6.156 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	<i>s</i>	The string to operate on.
in	<i>match</i>	The regex match object to find text to replace.
in	<i>repl</i>	The <a href="#">ColorArg</a> to produce escape-codes to replace with. <a href="#">ColorArg_free()</a> 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](#)

0.5.2.6.157 `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	<code>s</code>	The string to operate on.
in	<code>match</code>	The regex match object to find text to replace.
in	<code>repl</code>	The <a href="#">ColorResult</a> to produce escape-codes to replace with. <a href="#">ColorResult_free()</a> 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](#)

0.5.2.6.158 `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	<code>s</code>	The string to operate on.
in	<code>match</code>	The regex match object to find text to replace.
in	<code>repl</code>	The <a href="#">ColorText</a> to produce text/escape-codes to replace with. <a href="#">ColorText_free()</a> 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](#)

## 0.5.2.6.159 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	<i>ref</i>	The string to use for offset references. Can be <i>target</i> . Set this to the source string if <i>target</i> has not been filled yet. If <i>target</i> has been filled, you may use <i>target</i> for both <i>ref</i> and <i>target</i> .
out	<i>target</i>	The string to modify. Must have room for the resulting string.
in	<i>match</i>	The regex match object to find text to replace.
in	<i>repl</i>	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()`.

0.5.2.6.160 `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	<code>s</code>	The string to operate on.
in	<code>matches</code>	Regex match objects to find text to replace. The array must have <code>NULL</code> as the last member.
in	<code>repl</code>	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_ColorResult()`, `colr_str_replace_re_matches_ColorText()`, and `colr_str_replace_re_pat_all()`.

0.5.2.6.161 `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	<code>s</code>	The string to operate on.
in	<code>matches</code>	The regex match objects to find text to replace.
in	<code>repl</code>	The <a href="#">ColorArg</a> to produce escape-codes to replace with. <a href="#">ColorArg_free()</a> 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](#)

0.5.2.6.162 `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	<code>s</code>	The string to operate on.
in	<code>matches</code>	The regex match objects to find text to replace.
in	<code>repl</code>	The <a href="#">ColorResult</a> to produce escape-codes to replace with. <a href="#">ColorResult_free()</a> 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](#)

0.5.2.6.163 `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	<code>s</code>	The string to operate on.
in	<code>matches</code>	The regex match objects to find text to replace.
in	<code>repl</code>	The <code>ColorText</code> to produce text/escape-codes to replace with. <code>ColorText_free()</code> 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](#)

#### 0.5.2.6.164 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	<code>s</code>	The string to operate on.
in	<code>repattern</code>	The regex pattern to match ( <code>regex_t*</code> ).
in	<code>repl</code>	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()`.

## 0.5.2.6.165 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	<i>s</i>	The string to operate on.
in	<i>repattern</i>	The regex pattern to match ( <i>regex_t*</i> ).
in	<i>repl</i>	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()`.

0.5.2.6.166 `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	<code>s</code>	The string to operate on.
in	<code>repattern</code>	The regex pattern to match ( <code>regex_t*</code> ).
in	<code>repl</code>	The <a href="#">ColorArg</a> to produce escape-codes to replace with. <a href="#">ColorArg_free()</a> 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](#)

0.5.2.6.167 `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	<code>s</code>	The string to operate on.
in	<code>repattern</code>	The regex pattern to match ( <code>regex_t*</code> ).
in	<code>repl</code>	The <a href="#">ColorResult</a> to produce escape-codes to replace with. <a href="#">ColorResult_free()</a> 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](#)

## 0.5.2.6.168 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	<i>s</i>	The string to operate on.
in	<i>repattern</i>	The regex pattern to match (regex_t*).
in	<i>repl</i>	The <a href="#">ColorText</a> to produce text/escape-codes to replace with. <a href="#">ColorText_free()</a> 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](#)

## 0.5.2.6.169 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	<i>s</i>	The string to operate on.
in	<i>repattern</i>	The regex pattern to match ( <code>regex_t*</code> ).
in	<i>repl</i>	The <a href="#">ColorArg</a> to produce escape-codes to replace with. <a href="#">ColorArg_free()</a> 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](#)

## 0.5.2.6.170 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	<i>s</i>	The string to operate on.
in	<i>repattern</i>	The regex pattern to match ( <code>regex_t*</code> ).
in	<i>repl</i>	The <a href="#">ColorResult</a> to produce escape-codes to replace with. <a href="#">ColorResult_free()</a> 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](#)

0.5.2.6.171 `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	<code>s</code>	The string to operate on.
in	<code>repattern</code>	The regex pattern to match ( <code>regex_t*</code> ).
in	<code>repl</code>	The <a href="#">ColorText</a> to produce text/escape-codes to replace with. <a href="#">ColorText_free()</a> 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](#)

0.5.2.6.172 `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	<code>s</code>	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()`.

0.5.2.6.173 `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	<code>s</code>	The string to justify. Input <i>must be null-terminated</i> .
in	<code>width</code>	The overall width for the resulting string. If set to '0', the terminal width will be used from <a href="#">colr_term_size()</a> .
in	<code>padchar</code>	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()`.

## 0.5.2.6.174 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	<i>s</i>	The string to check.
in	<i>prefix</i>	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.

0.5.2.6.175 `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	<i>s</i>	The string to strip escape codes from. Input <i>must be null-terminated</i> .
----	----------	--

## 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()`.

0.5.2.6.176 `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. <i>Must be null-terminated.</i>
----	---	---

## 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().

## 0.5.2.6.177 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().

## 0.5.2.6.178 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.

0.5.2.6.179 `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()`.

0.5.2.6.180 `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()`.

0.5.2.6.181 `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()`.

0.5.2.6.182 `ExtendedValue_eq()`

```
bool ExtendedValue_eq (
    ExtendedValue a,
    ExtendedValue b )
```

Compares two `ExtendedValues`.

This is used to implement `colr_eq()`.

Parameters

in	<i>a</i>	The first <code>ExtendedValue</code> to compare.
in	<i>b</i>	The second <code>ExtendedValue</code> to compare.

Returns

true if they are equal, otherwise false.

See also

[ExtendedValue](#)

0.5.2.6.183 `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	<i>bval</i>	BasicValue to convert.
----	-------------	------------------------

## Returns

An ExtendedValue 0–15 on success, otherwise EXT\_INVALID.

## See also

[ExtendedValue](#)

## 0.5.2.6.184 ExtendedValue\_from\_esc()

```
int ExtendedValue_from_esc (
    const char * s )
```

Convert an escape-code string (char\*) to an ExtendedValue.

## Parameters

in	<i>s</i>	Escape-code string. <i>Must be null-terminated.</i>
----	----------	--

## Return values

<i>An</i>	integer in the range 0–255 on success.
<i>EXT_INVALID</i>	on error (or if <i>s</i> is NULL).
<i>EXT_INVALID_RANGE</i>	if the code number was outside of the range 0–255.

## See also

[ExtendedValue](#)

## 0.5.2.6.185 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	<i>hexstr</i>	Hex string to convert.
----	---------------	------------------------

## Returns

A value between 0 and 255 on success.

## Return values

<i>COLOR_INVALID</i>	on error or bad values.
----------------------	-------------------------

## See also

[ExtendedValue](#)

Referenced by `ExtendedValue_from_hex_default()`, and `ExtendedValue_from_str()`.

0.5.2.6.186 `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	<i>hexstr</i>	Hex string to convert.
in	<i>default_value</i>	<code>ExtendedValue</code> to use for bad hex strings.

## Returns

An `ExtendedValue` on success, or `default_value` on error.

See also

[ExtendedValue](#)  
[ExtendedValue\\_from\\_hex](#)

#### 0.5.2.6.187 ExtendedValue\_from\_RGB()

```
ExtendedValue ExtendedValue_from_RGB (
    RGB rgb )
```

Convert an [RGB](#) value into the closest matching [ExtendedValue](#).

Parameters

in	<i>rgb</i>	<a href="#">RGB</a> 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\(\)](#).

#### 0.5.2.6.188 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	<i>arg</i>	Color name to find the <a href="#">ExtendedValue</a> for.
----	------------	---

## Returns

A value between 0 and 255 on success.

## Return values

<i>EXT_INVALID</i>	on error or bad values.
<i>EXT_INVALID_RANGE</i>	if the number was outside of the range 0-255.

## See also

[ExtendedValue](#)

## 0.5.2.6.189 ExtendedValue\_is\_invalid()

```
bool ExtendedValue_is_invalid (  
    int eval )
```

Determines whether an integer is an invalid ExtendedValue.

## Parameters

in	<i>eval</i>	A number to check.
----	-------------	--------------------

## Returns

true if the value is considered invalid, otherwise false.

## See also

[ExtendedValue](#)

## 0.5.2.6.190 ExtendedValue\_is\_valid()

```
bool ExtendedValue_is_valid (  
    int eval )
```

Determines whether an integer is a valid ExtendedValue.

## Parameters

in	<i>eval</i>	A number to check.
----	-------------	--------------------

## Returns

true if the value is considered valid, otherwise false.

## See also

[ExtendedValue](#)

## 0.5.2.6.191 ExtendedValue\_repr()

```
char* ExtendedValue_repr (
    int eval )
```

Creates a string (char\*) representation of a ExtendedValue.

## Parameters

in	<i>eval</i>	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](#)

## 0.5.2.6.192 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	<i>eval</i>	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](#)

#### 0.5.2.6.193 `format_bg()`

```
void format_bg (
    char * out,
    BasicValue value )
```

Create an escape code for a background color.

Parameters

out	<i>out</i>	Memory allocated for the escape code string. <i>Must have enough room for CODEX_LEN.</i>
in	<i>value</i>	<a href="#">BasicValue</a> value to use for background.

#### 0.5.2.6.194 `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	<i>out</i>	Memory allocated for the escape code string. <i>Must have enough room for CODE_RGB_LEN.</i>
in	<i>rgb</i>	<a href="#">RGB</a> struct to get red, blue, and green values from.

Referenced by `_rainbow()`, and `rainbow_bg()`.

#### 0.5.2.6.195 `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	<i>out</i>	Memory allocated for the escape code string.
in	<i>rgb</i>	Pointer to an <a href="#">RGB</a> struct.

Referenced by `_rainbow()`, and `rainbow_bg_term()`.

0.5.2.6.196 `format_bgx()`

```
void format_bgx (
    char * out,
    unsigned char num )
```

Create an escape code for an extended background color.

## Parameters

out	<i>out</i>	Memory allocated for the escape code string. <i>Must have enough room for CODEX_LEN.</i>
in	<i>num</i>	Value to use for background.

Referenced by `format_bg_RGB_term()`.

0.5.2.6.197 `format_fg()`

```
void format_fg (
    char * out,
    BasicValue value )
```

Create an escape code for a fore color.

## Parameters

out	<i>out</i>	Memory allocated for the escape code string. <i>Must have enough room for CODEX_LEN.</i>
in	<i>value</i>	<a href="#">BasicValue</a> value to use for fore.

0.5.2.6.198 `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	<i>out</i>	Memory allocated for the escape code string.
in	<i>rgb</i>	Pointer to an <a href="#">RGB</a> struct.

Referenced by `rainbow_fg()`.

0.5.2.6.199 `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	<i>out</i>	Memory allocated for the escape code string.
in	<i>rgb</i>	Pointer to an <a href="#">RGB</a> struct.

Referenced by `rainbow_fg_term()`.

0.5.2.6.200 `format_fgx()`

```
void format_fgx (  
    char * out,  
    unsigned char num )
```

Create an escape code for an extended fore color.

## Parameters

out	<i>out</i>	Memory allocated for the escape code string. <i>Must have enough room for CODEX_LEN.</i>
in	<i>num</i>	Value to use for fore.

Referenced by `format_fg_RGB_term()`.

0.5.2.6.201 `format_style()`

```
void format_style (  
    char * out,  
    StyleValue style )
```

Create an escape code for a style.

## Parameters

out	<i>out</i>	Memory allocated for the escape code string. <i>Must have enough room for STYLE_LEN.</i>
in	<i>style</i>	StyleValue value to use for style.

## 0.5.2.6.202 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	<i>s</i>	The string to colorize. Input <i>must be null-terminated</i> .
in	<i>freq</i>	Frequency ("tightness") for the colors.
in	<i>offset</i>	Starting offset in the rainbow.
in	<i>spread</i>	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.*

## 0.5.2.6.203 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	<i>s</i>	The string to colorize. Input <i>must be null-terminated</i> .
in	<i>freq</i>	Frequency ("tightness") for the colors.
in	<i>offset</i>	Starting offset in the rainbow.
in	<i>spread</i>	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.*

## 0.5.2.6.204 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	<i>s</i>	The string to colorize. Input <i>must be null-terminated</i> .
in	<i>freq</i>	Frequency ("tightness") for the colors.
in	<i>offset</i>	Starting offset in the rainbow.
in	<i>spread</i>	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.*

## 0.5.2.6.205 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	<i>s</i>	The string to colorize. Input <i>must be null-terminated</i> .
in	<i>freq</i>	Frequency ("tightness") for the colors.
in	<i>offset</i>	Starting offset in the rainbow.
in	<i>spread</i>	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.*

## 0.5.2.6.206 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	<i>freq</i>	Frequency ("tightness") of the colors.
in	<i>offset</i>	Starting offset in the rainbow.

## Returns

An [RGB](#) value with the next "step" in the "rainbow".

Referenced by `_rainbow()`.



## 0.5.2.6.207 RGB\_average()

```
unsigned char RGB_average (
    RGB rgb )
```

Return the average for an [RGB](#) value.

This is also it's "grayscale" value.

Parameters

in	<i>rgb</i>	The <a href="#">RGB</a> value to get the average for.
----	------------	---

Returns

A value between 0–255.

See also

[RGB](#)

Referenced by RGB\_grayscale().

## 0.5.2.6.208 RGB\_eq()

```
bool RGB_eq (
    RGB a,
    RGB b )
```

Compare two [RGB](#) structs.

Parameters

in	<i>a</i>	First <a href="#">RGB</a> value to check.
in	<i>b</i>	Second <a href="#">RGB</a> 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().

## 0.5.2.6.209 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	<i>bval</i>	A BasicValue to get the <a href="#">RGB</a> value for.
----	-------------	--

## Returns

An [RGB](#) value that matches the BasicValue's color.

## See also

[RGB](#)

## 0.5.2.6.210 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	<i>s</i>	Escape-code string. <i>Must be null-terminated.</i>
out	<i>rgb</i>	Pointer to an <a href="#">RGB</a> struct to fill in the values for.

## Return values

<code>&lt;tt&gt;0&lt;/tt&gt;</code>	on success, with <code>rgb</code> filled with values.
<code>COLOR_INVALID</code>	on error (or if <code>s</code> is NULL).
<code>COLOR_INVALID_RANGE</code>	if any code numbers were outside of the range 0–255.

## See also

[RGB](#)

## 0.5.2.6.211 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	<i>eval</i>	An ExtendedValue to get the <a href="#">RGB</a> value for.
----	-------------	--

Returns

An [RGB](#) value from `ext2rgb_map[]`.

See also

[RGB](#)

## 0.5.2.6.212 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	<i>hexstr</i>	String to check for hex values. Input <i>must be null-terminated</i> .
out	<i>rgb</i>	Pointer to an <a href="#">RGB</a> struct to fill in the values for.

Return values

0	on success, with <i>rgb</i> filled with the values.
<i>COLOR_INVALID</i>	for non-hex strings.

See also

[RGB](#)

Referenced by `ExtendedValue_from_hex()`, `RGB_from_hex_default()`, and `RGB_from_str()`.

#### 0.5.2.6.213 `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	<i>hexstr</i>	String to check for <a href="#">RGB</a> values. Input <i>must be null-terminated</i> .
out	<i>default_value</i>	An <a href="#">RGB</a> value to use when errors occur.

Returns

A valid [RGB](#) value on success, or `default_value` on error.

See also

[RGB](#)  
[hex](#)

#### 0.5.2.6.214 `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	<i>arg</i>	String to check for <a href="#">RGB</a> values. Input <i>must be null-terminated</i> .
out	<i>rgb</i>	Pointer to an <a href="#">RGB</a> struct to fill in the values for.

## Return values

0	on success, with <code>rgb</code> filled with the values.
<code>COLOR_INVALID</code>	for non-rgb strings.
<code>COLOR_INVALID_RANGE</code>	for rgb values outside of 0-255.

See also

[RGB](#)

0.5.2.6.215 `RGB_grayscale()`

```
RGB RGB_grayscale (  
    RGB rgb )
```

Return a grayscale version of an [RGB](#) value.

## Parameters

in	<i>rgb</i>	The <a href="#">RGB</a> value to convert.
----	------------	---

## Returns

A grayscale [RGB](#) value.

See also

[RGB](#)

## 0.5.2.6.216 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	<i>rgb</i>	The RGB value to invert.
----	------------	--------------------------

## Returns

An "inverted" RGB value.

## See also

RGB

## 0.5.2.6.217 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	<i>rgb</i>	The RGB value to convert.
----	------------	---------------------------

## Returns

Either `rgb(1, 1, 1)` or `rgb(255, 255, 255)`.

## See also

RGB

## 0.5.2.6.218 RGB\_repr()

```
char* RGB_repr (  
    RGB rgb )
```

Creates a string (char\*) representation for an RGB value.

Allocates memory for the string representation.

## Parameters

in	<i>rgb</i>	<a href="#">RGB</a> struct to get the representation for.
----	------------	---

## Returns

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

## See also

[RGB](#)

0.5.2.6.219 `RGB_to_hex()`

```
char* RGB_to_hex (
    RGB rgb )
```

Converts an [RGB](#) value into a hex string (char\*).

## Parameters

in	<i>rgb</i>	<a href="#">RGB</a> 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](#)

0.5.2.6.220 `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	<i>rgb</i>	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](#)

## 0.5.2.6.221 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	<i>rgb</i>	RGB to convert.
----	------------	-----------------

## Returns

A new [RGB](#) with values close to a terminal code color.

## See also

[RGB](#)

Referenced by `ExtendedValue_from_RGB()`.

## 0.5.2.6.222 StyleValue\_eq()

```
bool StyleValue_eq (  
    StyleValue a,  
    StyleValue b )
```

Compares two StyleValues.

This is used to implement `colr_eq()`.



## Parameters

in	<i>a</i>	The first StyleValue to compare.
in	<i>b</i>	The second StyleValue to compare.

## Returns

true if they are equal, otherwise false.

See also

[StyleValue](#)

## 0.5.2.6.223 StyleValue\_from\_esc()

```
StyleValue StyleValue_from_esc (  
    const char * s )
```

Convert an escape-code string (char\*) to an actual StyleValue enum value.

## Parameters

in	<i>s</i>	Escape-code string. <i>Must be null-terminated.</i>
----	----------	--

## Return values

<i>StyleValue</i>	value on success.
<i>STYLE_INVALID</i>	on error (or if <i>s</i> is NULL).
<i>STYLE_INVALID_RANGE</i>	if the code number was outside of the range 0–255.

See also

[StyleValue](#)

## 0.5.2.6.224 StyleValue\_from\_str()

```
StyleValue StyleValue_from_str (  
    const char * arg )
```

Convert a named argument to actual StyleValue enum value.

## Parameters

in	<i>arg</i>	Style name to convert into a StyleValue.
----	------------	--

## Returns

A usable StyleValue value on success, or STYLE\_INVALID on error.

## See also

[StyleValue](#)

## 0.5.2.6.225 StyleValue\_is\_invalid()

```
bool StyleValue_is_invalid (  
    StyleValue sval )
```

Determines whether a StyleValue is invalid.

## Parameters

in	<i>sval</i>	A StyleValue to check.
----	-------------	------------------------

## Returns

true if the value is considered invalid, otherwise false.

## See also

[StyleValue](#)

## 0.5.2.6.226 StyleValue\_is\_valid()

```
bool StyleValue_is_valid (  
    StyleValue sval )
```

Determines whether a StyleValue is valid.

## Parameters

in	<i>sval</i>	A StyleValue to check.
----	-------------	------------------------

## Returns

true if the value is considered valid, otherwise false.

## See also

[StyleValue](#)

## 0.5.2.6.227 StyleValue\_repr()

```
char* StyleValue_repr (
    StyleValue sval )
```

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

## Parameters

in	<i>sval</i>	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](#)

## 0.5.2.6.228 StyleValue\_to\_str()

```
char* StyleValue_to_str (
    StyleValue sval )
```

Create a human-friendly string (char\*) representation for a StyleValue.

## Parameters

in	<i>sval</i>	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](#)

#### 0.5.2.6.229 TermSize\_repr()

```
char* TermSize_repr (
    TermSize ts )
```

Create a string (char\*) representation for a [TermSize](#).

Parameters

in	ts	<a href="#">TermSize</a> 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](#)

#### 0.5.2.7 Variable Documentation

##### 0.5.2.7.1 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().

## 0.6 Example Documentation

### 0.6.1 back\_example.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* colored = colr_cat(
    Colr("This is red.\n", back(RED)),
    Colr("This is also red.\n", fore("white"), back("red")),
    "This is not."
);
if (!colored) return 1;
printf("%s\n", colored);
free(colored);
}

```

## 0.6.2 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* colored = colr(mine, fore(BLUE), back(WHITE));
    if (!colored) return EXIT_FAILURE;
    printf("%s\n", colored);
    // Your string is still good:
    printf("%s\n", mine);

    char* appended = colr_cat(colored, "...still here.");
    if (!appended) return EXIT_FAILURE;
    printf("%s\n", appended);
    // The Colr-allocated string is still good:

```

```

printf("%s\n", colored);

// 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(colored),
    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).
*/
}

```

### 0.6.3 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 *colored = 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 colored, 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)), "?"));
}

```

## 0.6.4 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 separate 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 colored 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);
}

```

### 0.6.5 colr\_join\_example.c

```

#include "colr.h"

int main(void) {
    /*
        You can join things by a plain string or a colored string.

        For the pieces, the order/number of arguments does not matter.
        colr_join() accepts ColorArgs, ColorResults, ColorTexts, and strings (char*).
    */
    char* colored = 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 colored piece of text on it's own line:
    printf("%s\n", colored);
    free(colored);

    /*
        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 colored " <--> ".
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]);
}

```

## 0.6.6 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* myallocated = NULL;
if (colr_asprintf(&myallocated, "This: %R", Colr("Hah!", fore("dimgrey")))) < 1) {

```

```

        // Failed to allocate.
        return EXIT_FAILURE;
    }
    puts(myallocated);
    free(myallocated);
}

```

### 0.6.7 colr\_replace\_all\_example.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;
}

```

## 0.6.8 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;
}

```

## 0.6.9 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;
}

```



## 0.6.10 colr\_replace\_re\_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 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;
}

```

### 0.6.11 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);
}

```

### 0.6.12 simple\_example.c

```

#include "colr.h"

int main(int argc, char** argv) {
    // Print-related macros, using Colr() to build colored 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 colored string.
free(warning_label);

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

// Or colorize an existing string by replacing a regex pattern.
colored = colr_replace_re(
    logtext,
    "\\[[\\w+\\]]",
    Colr_join(
        Colr("ok", style(BRIGHT)),
        "(",
        ")"
    ),
    REG_EXTENDED
);
if (!colored) return EXIT_FAILURE;
puts(colored);
free(colored);

// 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* coloredall = colr_replace_all(
    logtext2,
    "warning",
    Colr("WARNING", fore(YELLOW))
);
// Failed to allocate for new string?
if (!coloredall) return EXIT_FAILURE;
puts(coloredall);
// You have to free the resulting string.
free(coloredall);
}

```

### 0.6.13 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

- `_colr_free`
  - `colr.c`, [28](#)
  - `colr.h`, [229](#)
- `_colr_is_last_arg`
  - `colr.c`, [28](#)
  - `colr.h`, [229](#)
- `_colr_join`
  - `colr.c`, [29](#)
  - `colr.h`, [229](#)
- `_colr_join_array_length`
  - `colr.c`, [29](#)
  - `colr.h`, [230](#)
- `_colr_join_arrayn_size`
  - `colr.c`, [30](#)
  - `colr.h`, [231](#)
- `_colr_join_size`
  - `colr.c`, [31](#)
  - `colr.h`, [231](#)
- `_colr_ptr_length`
  - `colr.c`, [31](#)
  - `colr.h`, [232](#)
- `_colr_ptr_repr`
  - `colr.c`, [32](#)
  - `colr.h`, [232](#)
- `_colr_ptr_to_str`
  - `colr.c`, [32](#)
  - `colr.h`, [233](#)
- `_rainbow`
  - `colr.c`, [33](#)
  - `colr.h`, [233](#)
- `alloc_basic`
  - `colr.h`, [175](#)
- `alloc_extended`
  - `colr.h`, [176](#)
- `alloc_rgb`
  - `colr.h`, [176](#)
- `alloc_style`
  - `colr.h`, [176](#)
- `ArgType_eq`
  - `colr.c`, [33](#)
  - `colr.h`, [234](#)
- `ArgType_repr`
  - `colr.c`, [34](#)
  - `colr.h`, [235](#)
- `ArgType_to_str`
  - `colr.c`, [34](#)
  - `colr.h`, [235](#)
- `asprintf_or_return`
  - `colr.h`, [176](#)
- `back`
  - `colr.h`, [177](#)
- `back_arg`
  - `colr.h`, [177](#)
- `back_str`
  - `colr.h`, [178](#)
- `back_str_static`
  - `colr.h`, [179](#)
- `basic`
  - `colr.h`, [180](#)
- `basic_names`
  - `colr.c`, [150](#)
- `BasicInfo`, [170](#)
- `BasicValue`
  - `colr.h`, [228](#)
- `BasicValue_eq`
  - `colr.c`, [35](#)
  - `colr.h`, [236](#)
- `BasicValue_from_esc`
  - `colr.c`, [35](#)
  - `colr.h`, [236](#)
- `BasicValue_from_str`
  - `colr.c`, [36](#)
  - `colr.h`, [237](#)
- `BasicValue_is_invalid`
  - `colr.c`, [36](#)
  - `colr.h`, [237](#)
- `BasicValue_is_valid`
  - `colr.c`, [37](#)
  - `colr.h`, [237](#)
- `BasicValue_repr`
  - `colr.c`, [37](#)
  - `colr.h`, [238](#)
- `BasicValue_to_ansi`
  - `colr.c`, [38](#)
  - `colr.h`, [238](#)
- `BasicValue_to_str`
  - `colr.c`, [38](#)
  - `colr.h`, [239](#)
- `bool_colr_enum`
  - `colr.h`, [180](#)
- `CODE_ANY_LEN`
  - `colr.h`, [181](#)
- `CODE_LEN_MIN`
  - `colr.h`, [181](#)
- `CODE_LEN`
  - `colr.h`, [181](#)

CODE\_RGB\_LEN\_MIN  
     colr.h, [181](#)  
 CODEX\_LEN\_MIN  
     colr.h, [181](#)  
 COLOR\_LEN  
     colr.h, [182](#)  
 COLOR\_RGB\_LEN  
     colr.h, [183](#)  
 COLORARG\_MARKER  
     colr.h, [184](#)  
 COLR\_GNU  
     colr.h, [194](#)  
 color\_arg  
     colr.h, [182](#)  
 color\_name\_is\_invalid  
     colr.h, [182](#)  
 color\_name\_is\_valid  
     colr.h, [183](#)  
 color\_val  
     colr.h, [183](#)  
 ColorArg, [170](#)  
 ColorArg\_empty  
     colr.c, [39](#)  
     colr.h, [239](#)  
 ColorArg\_eq  
     colr.c, [39](#)  
     colr.h, [240](#)  
 ColorArg\_example  
     colr.c, [39](#)  
     colr.h, [240](#)  
 ColorArg\_free  
     colr.c, [40](#)  
     colr.h, [241](#)  
 ColorArg\_from\_BasicValue  
     colr.c, [40](#)  
     colr.h, [241](#)  
 ColorArg\_from\_ExtendedValue  
     colr.c, [42](#)  
     colr.h, [242](#)  
 ColorArg\_from\_RGB  
     colr.c, [42](#)  
     colr.h, [243](#)  
 ColorArg\_from\_StyleValue  
     colr.c, [43](#)  
     colr.h, [244](#)  
 ColorArg\_from\_esc  
     colr.c, [41](#)  
     colr.h, [242](#)  
 ColorArg\_from\_str  
     colr.c, [43](#)  
     colr.h, [243](#)  
 ColorArg\_from\_value  
     colr.c, [44](#)  
     colr.h, [244](#)  
 ColorArg\_is\_empty  
     colr.c, [44](#)  
     colr.h, [245](#)  
 ColorArg\_is\_invalid  
     colr.c, [45](#)  
     colr.h, [245](#)  
 ColorArg\_is\_ptr  
     colr.c, [45](#)  
     colr.h, [246](#)  
 ColorArg\_is\_valid  
     colr.c, [46](#)  
     colr.h, [246](#)  
 ColorArg\_length  
     colr.c, [46](#)  
     colr.h, [247](#)  
 ColorArg\_repr  
     colr.c, [47](#)  
     colr.h, [247](#)  
 ColorArg\_to\_esc  
     colr.c, [47](#)  
     colr.h, [248](#)  
 ColorArg\_to\_esc\_s  
     colr.c, [48](#)  
     colr.h, [248](#)  
 ColorArg\_to\_ptr  
     colr.c, [48](#)  
     colr.h, [249](#)  
 ColorArgs\_array\_free  
     colr.c, [49](#)  
     colr.h, [250](#)  
 ColorArgs\_array\_repr  
     colr.c, [49](#)  
     colr.h, [250](#)  
 ColorArgs\_from\_str  
     colr.c, [50](#)  
     colr.h, [250](#)  
 ColorJustify, [171](#)  
 ColorJustify\_empty  
     colr.c, [50](#)  
     colr.h, [251](#)  
 ColorJustify\_eq  
     colr.c, [51](#)  
     colr.h, [251](#)  
 ColorJustify\_is\_empty  
     colr.c, [51](#)  
     colr.h, [252](#)  
 ColorJustify\_new  
     colr.c, [52](#)  
     colr.h, [252](#)  
 ColorJustify\_repr  
     colr.c, [52](#)  
     colr.h, [253](#)  
 ColorJustifyMethod\_repr  
     colr.c, [53](#)  
     colr.h, [253](#)  
 ColorNameData, [171](#)  
 ColorResult, [172](#)  
 ColorResult\_empty  
     colr.c, [53](#)  
     colr.h, [254](#)  
 ColorResult\_eq  
     colr.c, [53](#)



- colr.h, [254](#)
- ColorResult\_free
  - colr.c, [54](#)
  - colr.h, [255](#)
- ColorResult\_is\_ptr
  - colr.c, [54](#)
  - colr.h, [255](#)
- ColorResult\_length
  - colr.c, [55](#)
  - colr.h, [256](#)
- ColorResult\_new
  - colr.c, [55](#)
  - colr.h, [256](#)
- ColorResult\_repr
  - colr.c, [56](#)
  - colr.h, [257](#)
- ColorResult\_to\_ptr
  - colr.c, [56](#)
  - colr.h, [257](#)
- ColorResult\_to\_str
  - colr.c, [57](#)
  - colr.h, [258](#)
- ColorStructMarker, [172](#)
- ColorStructMarker.bytes, [172](#)
- ColorText, [173](#)
- ColorText\_empty
  - colr.c, [58](#)
  - colr.h, [258](#)
- ColorText\_free
  - colr.c, [58](#)
  - colr.h, [258](#)
- ColorText\_free\_args
  - colr.c, [58](#)
  - colr.h, [259](#)
- ColorText\_from\_values
  - colr.c, [59](#)
  - colr.h, [259](#)
- ColorText\_has\_arg
  - colr.c, [59](#)
  - colr.h, [260](#)
- ColorText\_has\_args
  - colr.c, [60](#)
  - colr.h, [260](#)
- ColorText\_is\_empty
  - colr.c, [60](#)
  - colr.h, [261](#)
- ColorText\_is\_ptr
  - colr.c, [61](#)
  - colr.h, [261](#)
- ColorText\_length
  - colr.c, [61](#)
  - colr.h, [262](#)
- ColorText\_repr
  - colr.c, [62](#)
  - colr.h, [262](#)
- ColorText\_set\_just
  - colr.c, [62](#)
  - colr.h, [263](#)
- ColorText\_set\_values
  - colr.c, [63](#)
  - colr.h, [263](#)
- ColorText\_to\_ptr
  - colr.c, [63](#)
  - colr.h, [264](#)
- ColorText\_to\_str
  - colr.c, [64](#)
  - colr.h, [264](#)
- ColorType\_eq
  - colr.c, [64](#)
  - colr.h, [265](#)
- ColorType\_from\_str
  - colr.c, [65](#)
  - colr.h, [265](#)
- ColorType\_is\_invalid
  - colr.c, [66](#)
  - colr.h, [266](#)
- ColorType\_is\_valid
  - colr.c, [66](#)
  - colr.h, [267](#)
- ColorType\_repr
  - colr.c, [67](#)
  - colr.h, [267](#)
- ColorType\_to\_str
  - colr.c, [67](#)
  - colr.h, [267](#)
- ColorValue, [174](#)
- ColorValue\_empty
  - colr.c, [68](#)
  - colr.h, [268](#)
- ColorValue\_eq
  - colr.c, [68](#)
  - colr.h, [268](#)
- ColorValue\_example
  - colr.c, [69](#)
  - colr.h, [269](#)
- ColorValue\_from\_esc
  - colr.c, [69](#)
  - colr.h, [269](#)
- ColorValue\_from\_str
  - colr.c, [70](#)
  - colr.h, [270](#)
- ColorValue\_from\_value
  - colr.c, [70](#)
  - colr.h, [271](#)
- ColorValue\_has
  - colr.h, [184](#)
- ColorValue\_has\_BasicValue
  - colr.c, [71](#)
  - colr.h, [271](#)
- ColorValue\_has\_ExtendedValue
  - colr.c, [72](#)
  - colr.h, [272](#)
- ColorValue\_has\_RGB
  - colr.c, [72](#)
  - colr.h, [272](#)
- ColorValue\_has\_StyleValue

- colr.c, [72](#)
- colr.h, [273](#)
- ColorValue\_is\_empty
  - colr.c, [73](#)
  - colr.h, [273](#)
- ColorValue\_is\_invalid
  - colr.c, [73](#)
  - colr.h, [274](#)
- ColorValue\_is\_valid
  - colr.c, [74](#)
  - colr.h, [274](#)
- ColorValue\_length
  - colr.c, [74](#)
  - colr.h, [274](#)
- ColorValue\_repr
  - colr.c, [75](#)
  - colr.h, [275](#)
- ColorValue\_to\_esc
  - colr.c, [75](#)
  - colr.h, [275](#)
- ColorValue\_to\_esc\_s
  - colr.c, [76](#)
  - colr.h, [276](#)
- Colr
  - colr.h, [186](#)
- colr
  - colr.h, [187](#)
- colr.c
  - \_colr\_free, [28](#)
  - \_colr\_is\_last\_arg, [28](#)
  - \_colr\_join, [29](#)
  - \_colr\_join\_array\_length, [29](#)
  - \_colr\_join\_arrayn\_size, [30](#)
  - \_colr\_join\_size, [31](#)
  - \_colr\_ptr\_length, [31](#)
  - \_colr\_ptr\_repr, [32](#)
  - \_colr\_ptr\_to\_str, [32](#)
  - \_rainbow, [33](#)
  - ArgType\_eq, [33](#)
  - ArgType\_repr, [34](#)
  - ArgType\_to\_str, [34](#)
  - basic\_names, [150](#)
  - BasicValue\_eq, [35](#)
  - BasicValue\_from\_esc, [35](#)
  - BasicValue\_from\_str, [36](#)
  - BasicValue\_is\_invalid, [36](#)
  - BasicValue\_is\_valid, [37](#)
  - BasicValue\_repr, [37](#)
  - BasicValue\_to\_ansi, [38](#)
  - BasicValue\_to\_str, [38](#)
  - ColorArg\_empty, [39](#)
  - ColorArg\_eq, [39](#)
  - ColorArg\_example, [39](#)
  - ColorArg\_free, [40](#)
  - ColorArg\_from\_BasicValue, [40](#)
  - ColorArg\_from\_ExtendedValue, [42](#)
  - ColorArg\_from\_RGB, [42](#)
  - ColorArg\_from\_StyleValue, [43](#)
  - ColorArg\_from\_esc, [41](#)
  - ColorArg\_from\_str, [43](#)
  - ColorArg\_from\_value, [44](#)
  - ColorArg\_is\_empty, [44](#)
  - ColorArg\_is\_invalid, [45](#)
  - ColorArg\_is\_ptr, [45](#)
  - ColorArg\_is\_valid, [46](#)
  - ColorArg\_length, [46](#)
  - ColorArg\_repr, [47](#)
  - ColorArg\_to\_esc, [47](#)
  - ColorArg\_to\_esc\_s, [48](#)
  - ColorArg\_to\_ptr, [48](#)
  - ColorArgs\_array\_free, [49](#)
  - ColorArgs\_array\_repr, [49](#)
  - ColorArgs\_from\_str, [50](#)
  - ColorJustify\_empty, [50](#)
  - ColorJustify\_eq, [51](#)
  - ColorJustify\_is\_empty, [51](#)
  - ColorJustify\_new, [52](#)
  - ColorJustify\_repr, [52](#)
  - ColorJustifyMethod\_repr, [53](#)
  - ColorResult\_empty, [53](#)
  - ColorResult\_eq, [53](#)
  - ColorResult\_free, [54](#)
  - ColorResult\_is\_ptr, [54](#)
  - ColorResult\_length, [55](#)
  - ColorResult\_new, [55](#)
  - ColorResult\_repr, [56](#)
  - ColorResult\_to\_ptr, [56](#)
  - ColorResult\_to\_str, [57](#)
  - ColorText\_empty, [58](#)
  - ColorText\_free, [58](#)
  - ColorText\_free\_args, [58](#)
  - ColorText\_from\_values, [59](#)
  - ColorText\_has\_arg, [59](#)
  - ColorText\_has\_args, [60](#)
  - ColorText\_is\_empty, [60](#)
  - ColorText\_is\_ptr, [61](#)
  - ColorText\_length, [61](#)
  - ColorText\_repr, [62](#)
  - ColorText\_set\_just, [62](#)
  - ColorText\_set\_values, [63](#)
  - ColorText\_to\_ptr, [63](#)
  - ColorText\_to\_str, [64](#)
  - ColorType\_eq, [64](#)
  - ColorType\_from\_str, [65](#)
  - ColorType\_is\_invalid, [66](#)
  - ColorType\_is\_valid, [66](#)
  - ColorType\_repr, [67](#)
  - ColorType\_to\_str, [67](#)
  - ColorValue\_empty, [68](#)
  - ColorValue\_eq, [68](#)
  - ColorValue\_example, [69](#)
  - ColorValue\_from\_esc, [69](#)
  - ColorValue\_from\_str, [70](#)
  - ColorValue\_from\_value, [70](#)
  - ColorValue\_has\_BasicValue, [71](#)
  - ColorValue\_has\_ExtendedValue, [72](#)

ColorValue\_has\_RGB, 72  
ColorValue\_has\_StyleValue, 72  
ColorValue\_is\_empty, 73  
ColorValue\_is\_invalid, 73  
ColorValue\_is\_valid, 74  
ColorValue\_length, 74  
ColorValue\_repr, 75  
ColorValue\_to\_esc, 75  
ColorValue\_to\_esc\_s, 76  
colr\_alloc\_regmatch, 76  
colr\_append\_reset, 77  
colr\_char\_escape\_char, 77  
colr\_char\_in\_str, 78  
colr\_char\_is\_code\_end, 78  
colr\_char\_repr, 79  
colr\_char\_should\_escape, 79  
colr\_check\_marker, 80  
colr\_empty\_str, 80  
colr\_free\_re\_matches, 81  
colr\_join\_array, 81  
colr\_join\_arrayn, 82  
colr\_mb\_len, 82  
colr\_printf\_esc\_mod, 151  
colr\_printf\_handler, 83  
colr\_printf\_info, 84  
colr\_printf\_register, 84  
colr\_re\_matches, 85  
colr\_set\_locale, 85  
colr\_str\_array\_contains, 86  
colr\_str\_array\_free, 86  
colr\_str\_center, 86  
colr\_str\_char\_count, 87  
colr\_str\_char\_lcount, 88  
colr\_str\_chars\_lcount, 88  
colr\_str\_code\_count, 89  
colr\_str\_code\_len, 89  
colr\_str\_copy, 89  
colr\_str\_ends\_with, 90  
colr\_str\_get\_codes, 91  
colr\_str\_has\_codes, 91  
colr\_str\_hash, 92  
colr\_str\_is\_all, 92  
colr\_str\_is\_codes, 93  
colr\_str\_is\_digits, 93  
colr\_str\_ljust, 93  
colr\_str\_lower, 94  
colr\_str\_lstrip, 94  
colr\_str\_lstrip\_char, 95  
colr\_str\_lstrip\_chars, 95  
colr\_str\_mb\_len, 96  
colr\_str\_noncode\_len, 96  
colr\_str\_replace, 97  
colr\_str\_replace\_ColorArg, 101  
colr\_str\_replace\_ColorResult, 101  
colr\_str\_replace\_ColorText, 102  
colr\_str\_replace\_all, 98  
colr\_str\_replace\_all\_ColorArg, 98  
colr\_str\_replace\_all\_ColorResult, 99  
colr\_str\_replace\_all\_ColorText, 99  
colr\_str\_replace\_cnt, 100  
colr\_str\_replace\_re, 102  
colr\_str\_replace\_re\_ColorArg, 106  
colr\_str\_replace\_re\_ColorResult, 107  
colr\_str\_replace\_re\_ColorText, 107  
colr\_str\_replace\_re\_all, 103  
colr\_str\_replace\_re\_all\_ColorArg, 104  
colr\_str\_replace\_re\_all\_ColorResult, 105  
colr\_str\_replace\_re\_all\_ColorText, 105  
colr\_str\_replace\_re\_match, 108  
colr\_str\_replace\_re\_match\_ColorArg, 109  
colr\_str\_replace\_re\_match\_ColorResult, 109  
colr\_str\_replace\_re\_match\_ColorText, 110  
colr\_str\_replace\_re\_match\_i, 111  
colr\_str\_replace\_re\_matches, 111  
colr\_str\_replace\_re\_matches\_ColorArg, 112  
colr\_str\_replace\_re\_matches\_ColorResult, 113  
colr\_str\_replace\_re\_matches\_ColorText, 113  
colr\_str\_replace\_re\_pat, 114  
colr\_str\_replace\_re\_pat\_ColorArg, 117  
colr\_str\_replace\_re\_pat\_ColorResult, 118  
colr\_str\_replace\_re\_pat\_ColorText, 118  
colr\_str\_replace\_re\_pat\_all, 115  
colr\_str\_replace\_re\_pat\_all\_ColorArg, 115  
colr\_str\_replace\_re\_pat\_all\_ColorResult, 116  
colr\_str\_replace\_re\_pat\_all\_ColorText, 117  
colr\_str\_repr, 119  
colr\_str\_rjust, 120  
colr\_str\_starts\_with, 120  
colr\_str\_strip\_codes, 122  
colr\_str\_to\_lower, 122  
colr\_supports\_rgb, 123  
colr\_supports\_rgb\_static, 123  
colr\_term\_size, 123  
colr\_win\_size, 124  
colr\_win\_size\_env, 124  
ext2rgb\_map, 151  
extended\_names, 151  
ExtendedValue\_eq, 125  
ExtendedValue\_from\_BasicValue, 125  
ExtendedValue\_from\_RGB, 128  
ExtendedValue\_from\_esc, 126  
ExtendedValue\_from\_hex, 126  
ExtendedValue\_from\_hex\_default, 127  
ExtendedValue\_from\_str, 128  
ExtendedValue\_is\_invalid, 129  
ExtendedValue\_is\_valid, 129  
ExtendedValue\_repr, 130  
ExtendedValue\_to\_str, 130  
format\_bg, 131  
format\_bg\_RGB\_term, 131  
format\_bg\_RGB, 131  
format\_bgx, 132  
format\_fg, 132  
format\_fg\_RGB\_term, 133

- format\_fg\_RGB, 132
- format\_fgx, 133
- format\_style, 133
- RGB\_average, 139
- RGB\_eq, 139
- RGB\_from\_BasicValue, 139
- RGB\_from\_ExtendedValue, 140
- RGB\_from\_esc, 140
- RGB\_from\_hex, 141
- RGB\_from\_hex\_default, 142
- RGB\_from\_str, 142
- RGB\_grayscale, 143
- RGB\_inverted, 143
- RGB\_monochrome, 144
- RGB\_repr, 144
- RGB\_to\_hex, 145
- RGB\_to\_str, 145
- RGB\_to\_term\_RGB, 146
- rainbow\_bg, 135
- rainbow\_bg\_term, 135
- rainbow\_fg, 137
- rainbow\_fg\_term, 137
- rainbow\_step, 138
- style\_names, 151
- StyleValue\_eq, 146
- StyleValue\_from\_esc, 147
- StyleValue\_from\_str, 147
- StyleValue\_is\_invalid, 148
- StyleValue\_is\_valid, 148
- StyleValue\_repr, 149
- StyleValue\_to\_str, 149
- TermSize\_repr, 150
- colr.c(0.3.6), 17
- colr.h
  - \_colr\_free, 229
  - \_colr\_is\_last\_arg, 229
  - \_colr\_join, 229
  - \_colr\_join\_array\_length, 230
  - \_colr\_join\_arrayn\_size, 231
  - \_colr\_join\_size, 231
  - \_colr\_ptr\_length, 232
  - \_colr\_ptr\_repr, 232
  - \_colr\_ptr\_to\_str, 233
  - \_rainbow, 233
  - alloc\_basic, 175
  - alloc\_extended, 176
  - alloc\_rgb, 176
  - alloc\_style, 176
  - ArgType\_eq, 234
  - ArgType\_repr, 235
  - ArgType\_to\_str, 235
  - asprintf\_or\_return, 176
  - back, 177
  - back\_arg, 177
  - back\_str, 178
  - back\_str\_static, 179
  - basic, 180
  - BasicValue, 228
  - BasicValue\_eq, 236
  - BasicValue\_from\_esc, 236
  - BasicValue\_from\_str, 237
  - BasicValue\_is\_invalid, 237
  - BasicValue\_is\_valid, 237
  - BasicValue\_repr, 238
  - BasicValue\_to\_ansi, 238
  - BasicValue\_to\_str, 239
  - bool\_colr\_enum, 180
  - CODE\_ANY\_LEN, 181
  - CODE\_LEN\_MIN, 181
  - CODE\_LEN, 181
  - CODE\_RGB\_LEN\_MIN, 181
  - CODEX\_LEN\_MIN, 181
  - COLOR\_LEN, 182
  - COLOR\_RGB\_LEN, 183
  - COLORARG\_MARKER, 184
  - COLR\_GNU, 194
  - color\_arg, 182
  - color\_name\_is\_invalid, 182
  - color\_name\_is\_valid, 183
  - color\_val, 183
  - ColorArg\_empty, 239
  - ColorArg\_eq, 240
  - ColorArg\_example, 240
  - ColorArg\_free, 241
  - ColorArg\_from\_BasicValue, 241
  - ColorArg\_from\_ExtendedValue, 242
  - ColorArg\_from\_RGB, 243
  - ColorArg\_from\_StyleValue, 244
  - ColorArg\_from\_esc, 242
  - ColorArg\_from\_str, 243
  - ColorArg\_from\_value, 244
  - ColorArg\_is\_empty, 245
  - ColorArg\_is\_invalid, 245
  - ColorArg\_is\_ptr, 246
  - ColorArg\_is\_valid, 246
  - ColorArg\_length, 247
  - ColorArg\_repr, 247
  - ColorArg\_to\_esc, 248
  - ColorArg\_to\_esc\_s, 248
  - ColorArg\_to\_ptr, 249
  - ColorArgs\_array\_free, 250
  - ColorArgs\_array\_repr, 250
  - ColorArgs\_from\_str, 250
  - ColorJustify\_empty, 251
  - ColorJustify\_eq, 251
  - ColorJustify\_is\_empty, 252
  - ColorJustify\_new, 252
  - ColorJustify\_repr, 253
  - ColorJustifyMethod\_repr, 253
  - ColorResult\_empty, 254
  - ColorResult\_eq, 254
  - ColorResult\_free, 255
  - ColorResult\_is\_ptr, 255
  - ColorResult\_length, 256
  - ColorResult\_new, 256
  - ColorResult\_repr, 257

ColorResult\_to\_ptr, 257  
ColorResult\_to\_str, 258  
ColorText\_empty, 258  
ColorText\_free, 258  
ColorText\_free\_args, 259  
ColorText\_from\_values, 259  
ColorText\_has\_arg, 260  
ColorText\_has\_args, 260  
ColorText\_is\_empty, 261  
ColorText\_is\_ptr, 261  
ColorText\_length, 262  
ColorText\_repr, 262  
ColorText\_set\_just, 263  
ColorText\_set\_values, 263  
ColorText\_to\_ptr, 264  
ColorText\_to\_str, 264  
ColorType\_eq, 265  
ColorType\_from\_str, 265  
ColorType\_is\_invalid, 266  
ColorType\_is\_valid, 267  
ColorType\_repr, 267  
ColorType\_to\_str, 267  
ColorValue\_empty, 268  
ColorValue\_eq, 268  
ColorValue\_example, 269  
ColorValue\_from\_esc, 269  
ColorValue\_from\_str, 270  
ColorValue\_from\_value, 271  
ColorValue\_has, 184  
ColorValue\_has\_BasicValue, 271  
ColorValue\_has\_ExtendedValue, 272  
ColorValue\_has\_RGB, 272  
ColorValue\_has\_StyleValue, 273  
ColorValue\_is\_empty, 273  
ColorValue\_is\_invalid, 274  
ColorValue\_is\_valid, 274  
ColorValue\_length, 274  
ColorValue\_repr, 275  
ColorValue\_to\_esc, 275  
ColorValue\_to\_esc\_s, 276  
Colr, 186  
colr, 187  
colr\_alloc\_len, 187  
colr\_alloc\_regmatch, 277  
colr\_append\_reset, 277  
colr\_asprintf, 188  
Colr\_cat, 188  
colr\_cat, 189  
Colr\_center, 190  
Colr\_center\_char, 190  
colr\_char\_escape\_char, 277  
colr\_char\_in\_str, 278  
colr\_char\_is\_code\_end, 278  
colr\_char\_repr, 279  
colr\_char\_should\_escape, 279  
colr\_check\_marker, 280  
colr\_empty\_str, 281  
colr\_eq, 191  
colr\_example, 192  
colr\_fprintf, 192  
colr\_free, 194  
colr\_free\_re\_matches, 281  
colr\_is\_empty, 195  
colr\_is\_invalid, 195  
colr\_is\_valid, 196  
colr\_is\_valid\_mblen, 196  
colr\_istr\_either, 197  
colr\_istr\_eq, 197  
Colr\_join, 198  
colr\_join, 199  
colr\_join\_array, 281  
colr\_join\_arrayn, 282  
colr\_length, 199  
Colr\_ljust, 200  
Colr\_ljust\_char, 201  
colr\_max, 201  
colr\_mb\_len, 283  
colr\_print, 202  
colr\_printf, 202  
colr\_printf\_esc\_mod, 350  
colr\_printf\_handler, 283  
colr\_printf\_info, 284  
colr\_printf\_macro, 203  
colr\_printf\_register, 285  
colr\_puts, 203  
colr\_re\_matches, 285  
colr\_replace, 204  
colr\_replace\_all, 205  
colr\_replace\_re, 207  
colr\_replace\_re\_all, 208  
colr\_repr, 209  
Colr\_rjust, 211  
Colr\_rjust\_char, 211  
colr\_set\_locale, 285  
colr\_snprintf, 212  
colr\_sprintf, 213  
colr\_str\_array\_contains, 286  
colr\_str\_array\_free, 286  
colr\_str\_center, 286  
colr\_str\_char\_count, 287  
colr\_str\_char\_lcount, 288  
colr\_str\_chars\_lcount, 288  
colr\_str\_code\_count, 289  
colr\_str\_code\_len, 289  
colr\_str\_copy, 289  
colr\_str\_either, 213  
colr\_str\_ends\_with, 290  
colr\_str\_eq, 214  
colr\_str\_get\_codes, 291  
colr\_str\_has\_codes, 291  
colr\_str\_hash, 292  
colr\_str\_is\_all, 292  
colr\_str\_is\_codes, 293  
colr\_str\_is\_digits, 293  
colr\_str\_ljust, 293  
colr\_str\_lower, 294

colr\_str\_lstrip, 294  
 colr\_str\_lstrip\_char, 295  
 colr\_str\_lstrip\_chars, 295  
 colr\_str\_mb\_len, 296  
 colr\_str\_noncode\_len, 296  
 colr\_str\_replace, 297  
 colr\_str\_replace\_ColorArg, 301  
 colr\_str\_replace\_ColorResult, 301  
 colr\_str\_replace\_ColorText, 302  
 colr\_str\_replace\_all, 298  
 colr\_str\_replace\_all\_ColorArg, 298  
 colr\_str\_replace\_all\_ColorResult, 299  
 colr\_str\_replace\_all\_ColorText, 299  
 colr\_str\_replace\_cnt, 300  
 colr\_str\_replace\_re, 302  
 colr\_str\_replace\_re\_ColorArg, 306  
 colr\_str\_replace\_re\_ColorResult, 307  
 colr\_str\_replace\_re\_ColorText, 307  
 colr\_str\_replace\_re\_all, 303  
 colr\_str\_replace\_re\_all\_ColorArg, 304  
 colr\_str\_replace\_re\_all\_ColorResult, 305  
 colr\_str\_replace\_re\_all\_ColorText, 305  
 colr\_str\_replace\_re\_match, 308  
 colr\_str\_replace\_re\_match\_ColorArg, 309  
 colr\_str\_replace\_re\_match\_ColorResult, 309  
 colr\_str\_replace\_re\_match\_ColorText, 310  
 colr\_str\_replace\_re\_match\_i, 311  
 colr\_str\_replace\_re\_matches, 311  
 colr\_str\_replace\_re\_matches\_ColorArg, 312  
 colr\_str\_replace\_re\_matches\_ColorResult, 313  
 colr\_str\_replace\_re\_matches\_ColorText, 313  
 colr\_str\_replace\_re\_pat, 314  
 colr\_str\_replace\_re\_pat\_ColorArg, 317  
 colr\_str\_replace\_re\_pat\_ColorResult, 318  
 colr\_str\_replace\_re\_pat\_ColorText, 318  
 colr\_str\_replace\_re\_pat\_all, 315  
 colr\_str\_replace\_re\_pat\_all\_ColorArg, 315  
 colr\_str\_replace\_re\_pat\_all\_ColorResult, 316  
 colr\_str\_replace\_re\_pat\_all\_ColorText, 317  
 colr\_str\_repr, 319  
 colr\_str\_rjust, 320  
 colr\_str\_starts\_with, 320  
 colr\_str\_strip\_codes, 322  
 colr\_str\_to\_lower, 322  
 colr\_supports\_rgb, 323  
 colr\_supports\_rgb\_static, 323  
 colr\_term\_size, 323  
 colr\_to\_str, 214  
 colr\_win\_size, 324  
 colr\_win\_size\_env, 324  
 ColrResult, 216  
 Colra, 215  
 EXT\_INVALID\_RANGE, 218  
 EXT\_INVALID, 218  
 ext, 216  
 ext\_RGB, 219  
 ext\_hex, 217  
 ext\_hex\_or, 217  
 ext\_rgb, 219  
 ExtendedValue\_eq, 325  
 ExtendedValue\_from\_BasicValue, 325  
 ExtendedValue\_from\_RGB, 328  
 ExtendedValue\_from\_esc, 326  
 ExtendedValue\_from\_hex, 326  
 ExtendedValue\_from\_hex\_default, 327  
 ExtendedValue\_from\_str, 328  
 ExtendedValue\_is\_invalid, 329  
 ExtendedValue\_is\_valid, 329  
 ExtendedValue\_repr, 330  
 ExtendedValue\_to\_str, 330  
 fore, 220  
 fore\_arg, 221  
 fore\_str, 221  
 fore\_str\_static, 222  
 format\_bg, 331  
 format\_bg\_RGB\_term, 331  
 format\_bg\_RGB, 331  
 format\_bgx, 332  
 format\_fg, 332  
 format\_fg\_RGB\_term, 333  
 format\_fg\_RGB, 332  
 format\_fgx, 333  
 format\_style, 333  
 hex, 223  
 hex\_or, 223  
 if\_not\_asprintf, 224  
 RGB\_average, 339  
 RGB\_eq, 339  
 RGB\_fmter, 228  
 RGB\_from\_BasicValue, 339  
 RGB\_from\_ExtendedValue, 340  
 RGB\_from\_esc, 340  
 RGB\_from\_hex, 341  
 RGB\_from\_hex\_default, 342  
 RGB\_from\_str, 342  
 RGB\_grayscale, 343  
 RGB\_inverted, 343  
 RGB\_monochrome, 344  
 RGB\_repr, 344  
 RGB\_to\_hex, 345  
 RGB\_to\_str, 345  
 RGB\_to\_term\_RGB, 346  
 rainbow\_bg, 335  
 rainbow\_bg\_term, 335  
 rainbow\_fg, 337  
 rainbow\_fg\_term, 337  
 rainbow\_step, 338  
 rgb, 224  
 STYLE\_LEN\_MIN, 226  
 style, 225  
 style\_arg, 226  
 style\_str, 226  
 style\_str\_static, 227  
 StyleValue\_eq, 346

- StyleValue\_from\_esc, [347](#)
- StyleValue\_from\_str, [347](#)
- StyleValue\_is\_invalid, [348](#)
- StyleValue\_is\_valid, [348](#)
- StyleValue\_repr, [349](#)
- StyleValue\_to\_str, [349](#)
- TermSize\_repr, [350](#)
- while\_colr\_va\_arg, [228](#)
- colr.h(0.3.6), [152](#)
- colr\_alloc\_len
  - colr.h, [187](#)
- colr\_alloc\_regmatch
  - colr.c, [76](#)
  - colr.h, [277](#)
- colr\_append\_reset
  - colr.c, [77](#)
  - colr.h, [277](#)
- colr\_asprintf
  - colr.h, [188](#)
- Colr\_cat
  - colr.h, [188](#)
- colr\_cat
  - colr.h, [189](#)
- Colr\_center
  - colr.h, [190](#)
- Colr\_center\_char
  - colr.h, [190](#)
- colr\_char\_escape\_char
  - colr.c, [77](#)
  - colr.h, [277](#)
- colr\_char\_in\_str
  - colr.c, [78](#)
  - colr.h, [278](#)
- colr\_char\_is\_code\_end
  - colr.c, [78](#)
  - colr.h, [278](#)
- colr\_char\_repr
  - colr.c, [79](#)
  - colr.h, [279](#)
- colr\_char\_should\_escape
  - colr.c, [79](#)
  - colr.h, [279](#)
- colr\_check\_marker
  - colr.c, [80](#)
  - colr.h, [280](#)
- colr\_empty\_str
  - colr.c, [80](#)
  - colr.h, [281](#)
- colr\_eq
  - colr.h, [191](#)
- colr\_example
  - colr.h, [192](#)
- colr\_fprintf
  - colr.h, [192](#)
- colr\_free
  - colr.h, [194](#)
- colr\_free\_re\_matches
  - colr.c, [81](#)
- colr.h, [281](#)
- colr\_is\_empty
  - colr.h, [195](#)
- colr\_is\_invalid
  - colr.h, [195](#)
- colr\_is\_valid
  - colr.h, [196](#)
- colr\_is\_valid\_mblen
  - colr.h, [196](#)
- colr\_istr\_either
  - colr.h, [197](#)
- colr\_istr\_eq
  - colr.h, [197](#)
- Colr\_join
  - colr.h, [198](#)
- colr\_join
  - colr.h, [199](#)
- colr\_join\_array
  - colr.c, [81](#)
  - colr.h, [281](#)
- colr\_join\_arrayn
  - colr.c, [82](#)
  - colr.h, [282](#)
- colr\_length
  - colr.h, [199](#)
- Colr\_ljust
  - colr.h, [200](#)
- Colr\_ljust\_char
  - colr.h, [201](#)
- colr\_max
  - colr.h, [201](#)
- colr\_mb\_len
  - colr.c, [82](#)
  - colr.h, [283](#)
- colr\_print
  - colr.h, [202](#)
- colr\_printf
  - colr.h, [202](#)
- colr\_printf\_esc\_mod
  - colr.c, [151](#)
  - colr.h, [350](#)
- colr\_printf\_handler
  - colr.c, [83](#)
  - colr.h, [283](#)
- colr\_printf\_info
  - colr.c, [84](#)
  - colr.h, [284](#)
- colr\_printf\_macro
  - colr.h, [203](#)
- colr\_printf\_register
  - colr.c, [84](#)
  - colr.h, [285](#)
- colr\_puts
  - colr.h, [203](#)
- colr\_re\_matches
  - colr.c, [85](#)
  - colr.h, [285](#)
- colr\_replace



- colr.h, [204](#)
- colr\_replace\_all
  - colr.h, [205](#)
- colr\_replace\_re
  - colr.h, [207](#)
- colr\_replace\_re\_all
  - colr.h, [208](#)
- colr\_repr
  - colr.h, [209](#)
- Colr\_rjust
  - colr.h, [211](#)
- Colr\_rjust\_char
  - colr.h, [211](#)
- colr\_set\_locale
  - colr.c, [85](#)
  - colr.h, [285](#)
- colr\_snprintf
  - colr.h, [212](#)
- colr\_sprintf
  - colr.h, [213](#)
- colr\_str\_array\_contains
  - colr.c, [86](#)
  - colr.h, [286](#)
- colr\_str\_array\_free
  - colr.c, [86](#)
  - colr.h, [286](#)
- colr\_str\_center
  - colr.c, [86](#)
  - colr.h, [286](#)
- colr\_str\_char\_count
  - colr.c, [87](#)
  - colr.h, [287](#)
- colr\_str\_char\_lcount
  - colr.c, [88](#)
  - colr.h, [288](#)
- colr\_str\_chars\_lcount
  - colr.c, [88](#)
  - colr.h, [288](#)
- colr\_str\_code\_count
  - colr.c, [89](#)
  - colr.h, [289](#)
- colr\_str\_code\_len
  - colr.c, [89](#)
  - colr.h, [289](#)
- colr\_str\_copy
  - colr.c, [89](#)
  - colr.h, [289](#)
- colr\_str\_either
  - colr.h, [213](#)
- colr\_str\_ends\_with
  - colr.c, [90](#)
  - colr.h, [290](#)
- colr\_str\_eq
  - colr.h, [214](#)
- colr\_str\_get\_codes
  - colr.c, [91](#)
  - colr.h, [291](#)
- colr\_str\_has\_codes
  - colr.c, [91](#)
  - colr.h, [291](#)
- colr\_str\_hash
  - colr.c, [92](#)
  - colr.h, [292](#)
- colr\_str\_is\_all
  - colr.c, [92](#)
  - colr.h, [292](#)
- colr\_str\_is\_codes
  - colr.c, [93](#)
  - colr.h, [293](#)
- colr\_str\_is\_digits
  - colr.c, [93](#)
  - colr.h, [293](#)
- colr\_str\_ljust
  - colr.c, [93](#)
  - colr.h, [293](#)
- colr\_str\_lower
  - colr.c, [94](#)
  - colr.h, [294](#)
- colr\_str\_lstrip
  - colr.c, [94](#)
  - colr.h, [294](#)
- colr\_str\_lstrip\_char
  - colr.c, [95](#)
  - colr.h, [295](#)
- colr\_str\_lstrip\_chars
  - colr.c, [95](#)
  - colr.h, [295](#)
- colr\_str\_mb\_len
  - colr.c, [96](#)
  - colr.h, [296](#)
- colr\_str\_noncode\_len
  - colr.c, [96](#)
  - colr.h, [296](#)
- colr\_str\_replace
  - colr.c, [97](#)
  - colr.h, [297](#)
- colr\_str\_replace\_ColorArg
  - colr.c, [101](#)
  - colr.h, [301](#)
- colr\_str\_replace\_ColorResult
  - colr.c, [101](#)
  - colr.h, [301](#)
- colr\_str\_replace\_ColorText
  - colr.c, [102](#)
  - colr.h, [302](#)
- colr\_str\_replace\_all
  - colr.c, [98](#)
  - colr.h, [298](#)
- colr\_str\_replace\_all\_ColorArg
  - colr.c, [98](#)
  - colr.h, [298](#)
- colr\_str\_replace\_all\_ColorResult
  - colr.c, [99](#)
  - colr.h, [299](#)
- colr\_str\_replace\_all\_ColorText
  - colr.c, [99](#)



- colr.h, [299](#)
- colr\_str\_replace\_cnt
  - colr.c, [100](#)
  - colr.h, [300](#)
- colr\_str\_replace\_re
  - colr.c, [102](#)
  - colr.h, [302](#)
- colr\_str\_replace\_re\_ColorArg
  - colr.c, [106](#)
  - colr.h, [306](#)
- colr\_str\_replace\_re\_ColorResult
  - colr.c, [107](#)
  - colr.h, [307](#)
- colr\_str\_replace\_re\_ColorText
  - colr.c, [107](#)
  - colr.h, [307](#)
- colr\_str\_replace\_re\_all
  - colr.c, [103](#)
  - colr.h, [303](#)
- colr\_str\_replace\_re\_all\_ColorArg
  - colr.c, [104](#)
  - colr.h, [304](#)
- colr\_str\_replace\_re\_all\_ColorResult
  - colr.c, [105](#)
  - colr.h, [305](#)
- colr\_str\_replace\_re\_all\_ColorText
  - colr.c, [105](#)
  - colr.h, [305](#)
- colr\_str\_replace\_re\_match
  - colr.c, [108](#)
  - colr.h, [308](#)
- colr\_str\_replace\_re\_match\_ColorArg
  - colr.c, [109](#)
  - colr.h, [309](#)
- colr\_str\_replace\_re\_match\_ColorResult
  - colr.c, [109](#)
  - colr.h, [309](#)
- colr\_str\_replace\_re\_match\_ColorText
  - colr.c, [110](#)
  - colr.h, [310](#)
- colr\_str\_replace\_re\_match\_i
  - colr.c, [111](#)
  - colr.h, [311](#)
- colr\_str\_replace\_re\_matches
  - colr.c, [111](#)
  - colr.h, [311](#)
- colr\_str\_replace\_re\_matches\_ColorArg
  - colr.c, [112](#)
  - colr.h, [312](#)
- colr\_str\_replace\_re\_matches\_ColorResult
  - colr.c, [113](#)
  - colr.h, [313](#)
- colr\_str\_replace\_re\_matches\_ColorText
  - colr.c, [113](#)
  - colr.h, [313](#)
- colr\_str\_replace\_re\_pat
  - colr.c, [114](#)
  - colr.h, [314](#)
- colr\_str\_replace\_re\_pat\_ColorArg
  - colr.c, [117](#)
  - colr.h, [317](#)
- colr\_str\_replace\_re\_pat\_ColorResult
  - colr.c, [118](#)
  - colr.h, [318](#)
- colr\_str\_replace\_re\_pat\_ColorText
  - colr.c, [118](#)
  - colr.h, [318](#)
- colr\_str\_replace\_re\_pat\_all
  - colr.c, [115](#)
  - colr.h, [315](#)
- colr\_str\_replace\_re\_pat\_all\_ColorArg
  - colr.c, [115](#)
  - colr.h, [315](#)
- colr\_str\_replace\_re\_pat\_all\_ColorResult
  - colr.c, [116](#)
  - colr.h, [316](#)
- colr\_str\_replace\_re\_pat\_all\_ColorText
  - colr.c, [117](#)
  - colr.h, [317](#)
- colr\_str\_repr
  - colr.c, [119](#)
  - colr.h, [319](#)
- colr\_str\_rjust
  - colr.c, [120](#)
  - colr.h, [320](#)
- colr\_str\_starts\_with
  - colr.c, [120](#)
  - colr.h, [320](#)
- colr\_str\_strip\_codes
  - colr.c, [122](#)
  - colr.h, [322](#)
- colr\_str\_to\_lower
  - colr.c, [122](#)
  - colr.h, [322](#)
- colr\_supports\_rgb
  - colr.c, [123](#)
  - colr.h, [323](#)
- colr\_supports\_rgb\_static
  - colr.c, [123](#)
  - colr.h, [323](#)
- colr\_term\_size
  - colr.c, [123](#)
  - colr.h, [323](#)
- colr\_to\_str
  - colr.h, [214](#)
- colr\_win\_size
  - colr.c, [124](#)
  - colr.h, [324](#)
- colr\_win\_size\_env
  - colr.c, [124](#)
  - colr.h, [324](#)
- ColrResult
  - colr.h, [216](#)
- Colra
  - colr.h, [215](#)
- EXT\_INVALID\_RANGE

- colr.h, [218](#)
- EXT\_INVALID
  - colr.h, [218](#)
- ext
  - colr.h, [216](#)
- ext2rgb\_map
  - colr.c, [151](#)
- ext\_RGB
  - colr.h, [219](#)
- ext\_hex
  - colr.h, [217](#)
- ext\_hex\_or
  - colr.h, [217](#)
- ext\_rgb
  - colr.h, [219](#)
- extended\_names
  - colr.c, [151](#)
- ExtendedInfo, [174](#)
- ExtendedValue\_eq
  - colr.c, [125](#)
  - colr.h, [325](#)
- ExtendedValue\_from\_BasicValue
  - colr.c, [125](#)
  - colr.h, [325](#)
- ExtendedValue\_from\_RGB
  - colr.c, [128](#)
  - colr.h, [328](#)
- ExtendedValue\_from\_esc
  - colr.c, [126](#)
  - colr.h, [326](#)
- ExtendedValue\_from\_hex
  - colr.c, [126](#)
  - colr.h, [326](#)
- ExtendedValue\_from\_hex\_default
  - colr.c, [127](#)
  - colr.h, [327](#)
- ExtendedValue\_from\_str
  - colr.c, [128](#)
  - colr.h, [328](#)
- ExtendedValue\_is\_invalid
  - colr.c, [129](#)
  - colr.h, [329](#)
- ExtendedValue\_is\_valid
  - colr.c, [129](#)
  - colr.h, [329](#)
- ExtendedValue\_repr
  - colr.c, [130](#)
  - colr.h, [330](#)
- ExtendedValue\_to\_str
  - colr.c, [130](#)
  - colr.h, [330](#)
- fore
  - colr.h, [220](#)
- fore\_arg
  - colr.h, [221](#)
- fore\_str
  - colr.h, [221](#)
- fore\_str\_static
  - colr.h, [222](#)
- format\_bg
  - colr.c, [131](#)
  - colr.h, [331](#)
- format\_bg\_RGB\_term
  - colr.c, [131](#)
  - colr.h, [331](#)
- format\_bg\_RGB
  - colr.c, [131](#)
  - colr.h, [331](#)
- format\_bgx
  - colr.c, [132](#)
  - colr.h, [332](#)
- format\_fg
  - colr.c, [132](#)
  - colr.h, [332](#)
- format\_fg\_RGB\_term
  - colr.c, [133](#)
  - colr.h, [333](#)
- format\_fg\_RGB
  - colr.c, [132](#)
  - colr.h, [332](#)
- format\_fgx
  - colr.c, [133](#)
  - colr.h, [333](#)
- format\_style
  - colr.c, [133](#)
  - colr.h, [333](#)
- hex
  - colr.h, [223](#)
- hex\_or
  - colr.h, [223](#)
- if\_not\_asprintf
  - colr.h, [224](#)
- RGB\_average
  - colr.c, [139](#)
  - colr.h, [339](#)
- RGB\_eq
  - colr.c, [139](#)
  - colr.h, [339](#)
- RGB\_fmter
  - colr.h, [228](#)
- RGB\_from\_BasicValue
  - colr.c, [139](#)
  - colr.h, [339](#)
- RGB\_from\_ExtendedValue
  - colr.c, [140](#)
  - colr.h, [340](#)
- RGB\_from\_esc
  - colr.c, [140](#)
  - colr.h, [340](#)
- RGB\_from\_hex
  - colr.c, [141](#)
  - colr.h, [341](#)
- RGB\_from\_hex\_default
  - colr.c, [142](#)

- colr.h, [342](#)
- RGB\_from\_str
  - colr.c, [142](#)
  - colr.h, [342](#)
- RGB\_grayscale
  - colr.c, [143](#)
  - colr.h, [343](#)
- RGB\_inverted
  - colr.c, [143](#)
  - colr.h, [343](#)
- RGB\_monochrome
  - colr.c, [144](#)
  - colr.h, [344](#)
- RGB\_repr
  - colr.c, [144](#)
  - colr.h, [344](#)
- RGB\_to\_hex
  - colr.c, [145](#)
  - colr.h, [345](#)
- RGB\_to\_str
  - colr.c, [145](#)
  - colr.h, [345](#)
- RGB\_to\_term\_RGB
  - colr.c, [146](#)
  - colr.h, [346](#)
- RGB, [175](#)
- rainbow\_bg
  - colr.c, [135](#)
  - colr.h, [335](#)
- rainbow\_bg\_term
  - colr.c, [135](#)
  - colr.h, [335](#)
- rainbow\_fg
  - colr.c, [137](#)
  - colr.h, [337](#)
- rainbow\_fg\_term
  - colr.c, [137](#)
  - colr.h, [337](#)
- rainbow\_step
  - colr.c, [138](#)
  - colr.h, [338](#)
- rgb
  - colr.h, [224](#)
- STYLE\_LEN\_MIN
  - colr.h, [226](#)
- style
  - colr.h, [225](#)
- style\_arg
  - colr.h, [226](#)
- style\_names
  - colr.c, [151](#)
- style\_str
  - colr.h, [226](#)
- style\_str\_static
  - colr.h, [227](#)
- StyleInfo, [175](#)
- StyleValue\_eq
  - colr.c, [146](#)
  - colr.h, [346](#)
- StyleValue\_from\_esc
  - colr.c, [147](#)
  - colr.h, [347](#)
- StyleValue\_from\_str
  - colr.c, [147](#)
  - colr.h, [347](#)
- StyleValue\_is\_invalid
  - colr.c, [148](#)
  - colr.h, [348](#)
- StyleValue\_is\_valid
  - colr.c, [148](#)
  - colr.h, [348](#)
- StyleValue\_repr
  - colr.c, [149](#)
  - colr.h, [349](#)
- StyleValue\_to\_str
  - colr.c, [149](#)
  - colr.h, [349](#)
- TermSize, [175](#)
- TermSize\_repr
  - colr.c, [150](#)
  - colr.h, [350](#)
- while\_colr\_va\_arg
  - colr.h, [228](#)