Mp

Generated by Doxygen 1.8.13

Contents

Chapter 1

Project Mp

Intro

- Succinctly, this is a fractal play pen.
- It is *not* a library.
- There is no ∗∗_API_∗∗
  - There are s lot of functions that might be of use
  - They may even one day be documented
- List its most useful/innovative/noteworthy features.
  - feature one
  - feature two
- State its goals/what problem(s) it solves.
  - Dust off my programming skills.
  - As much as possible work with what's out there. . .
  - Continue working with ∗fractals∗—probably the real point.
- Key concepts.
  - concept one
  - concept two
- This is now and always will be *alpha*.
- Does not include badges.

Core Technical Concepts/Inspiration

- Why does it exist?
- Frame your project for the potential user.
- Compare/contrast your project with other, similar projects so the user knows how it is different from those projects.
- Highlight the technical concepts that your project demonstrates or supports. Keep it very brief.

Getting Started/Requirements/Prerequisites/Dependencies

Include any essential instructions for:

- Getting it
- Installing It
- Configuring It
- Running it

Mini-Manual...

```
    printf( "%s v%s dated %s\n", Program, Version, Date );
    if ( c == 'h' || c == '?' ) {
        printf( "\n  Options:\n\n" );
        printf( "  --aa           requires filename as an argument -a\n" );
        printf( "  --bourke       requires number as an argument -b\n" );
        printf( "  --color        requires number as an argument -r\n" );
        printf( "  --config       requires filename as an argument -c\n" );
        printf( "  --diameter     requires real as an argument   -d\n" );
        printf( "  --escape       requires real as an argument   -e\n" );
        printf( "  --file         requires filename as an argument -f\n" );
        printf( "  --height       requires number as an argument -l\n" );
        printf( "  --help         no argument                    -h\n" );
        printf( "  --iteration    requires number as an argument -i\n" );
        printf( "  --json         requires filename as an argument -j\n" );
        printf( "       may be prefixed with W: to write parameters\n" );
        printf( "  --kolor        requires RGB spec as an argument -k\n" );
        printf( "       allows up to %d RGB color specifications\n", NAMES_SIZE );
        printf( "       which may be 'named' colors or braced {..}\n" );
        printf( "  --magnify      requires real as an argument   -m\n" );
        printf( "  --next         argument is optional number    -n\n" );
        printf( "  --old          requires number as argument    -o\n" );
        printf( "  --palette      requires filename as an argument -p\n" );
        printf( "  --semidiameter requires number as an argument -s\n" );
        printf( "       this is an alias for radius\n" );
        printf( "  --tweak        requires number as an argument -t\n" );
        printf( "  --version      no argument                    -v\n" );
        printf( "  --width        requires number as an argument -w\n" );
        printf( "  --x_center     requires real as an argument   -x\n" );
        printf( "  --y_center     requires real as an argument   -y\n" );
        printf( "\n  long options ('--' prefix) are incremental till unambiguous\n" );
        printf( "  short options ('-' prefix) are exact\n" );
        printf( "  Hideously enough, options with optional arguments,\n" );
        printf( "  take the form [-[-]]option=arg, no spaces.\n" );
    }
    exit( 0 );
}
```

Contributing

- Chris Thomasson
- Greg Harley

TODO

- Improve documentation
- Next steps
- Features planned
- Known bugs (shortlist)

Contact

- hsmyers@gmail.com
- http://www.sdragons.org/

License

- see file License.txt

Chapter 2

Todo List

File palette.c

Refactor code to eliminate duplication and near-duplication.

Chapter 3

Bug List

File cnames.c

No known bugs.

File elapsed.c

No known bugs.

File getopt.c

No known bugs.

File getopt.h

No known bugs.

File palette.c

No known bugs.

File palette.h

No known bugs.

File util.c

No known bugs.

File util.h

No known bugs.

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

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

Chapter 6

Data Structure Documentation

6.1 cJSON Struct Reference

Data Fields

- struct cJSON * **next**
- struct cJSON * **prev**
- struct cJSON * **child**
- int **type**
- char * **valuestring**
- int **valueint**
- double **valuedouble**
- char * **string**

6.1.1 Field Documentation

6.1.1.1 child

```
struct cJSON* cJSON::child
```

6.1.1.2 next

```
struct cJSON* cJSON::next
```

6.1.1.3 prev

```
struct cJSON* cJSON::prev
```

6.1.1.4 string

```
char* cJSON::string
```

6.1.1.5 type

```
int cJSON::type
```

6.1.1.6 valuedouble

```
double cJSON::valuedouble
```

6.1.1.7 valueint

```
int cJSON::valueint
```

6.1.1.8 valuestring

```
char* cJSON::valuestring
```

The documentation for this struct was generated from the following file:

- cJSON.h

6.2 cJSON_Hooks Struct Reference

Data Fields

- void *(* **malloc_fn**)(size_t sz)
- void(* **free_fn**)(void *ptr)

The documentation for this struct was generated from the following file:

- cJSON.h

6.3 Color128 Struct Reference

Data Fields

- __float128 **Zx**
- __float128 **Zy**
- __float128 **Cx**
- __float128 **Cy**
- __float128 **Zx2**
- __float128 **Zy2**
- __complex128 **Z**
- __complex128 **C**
- __float128 **colorPoly**
- __float128 **Exps**
- __float128 **DistanceMax**
- int **n**

6.3.1 Field Documentation

6.3.1.1 C

```
__complex128 Color128::C
```

6.3.1.2 colorPoly

```
__float128 Color128::colorPoly
```

6.3.1.3 Cx

```
__float128 Color128::Cx
```

6.3.1.4 Cy

```
__float128 Color128::Cy
```

6.3.1.5 DistanceMax

```
__float128 Color128::DistanceMax
```

6.3.1.6 Exps

```
__float128 Color128::Exps
```

6.3.1.7 n

```
int Color128::n
```

6.3.1.8 Z

```
__complex128 Color128::Z
```

6.3.1.9 Zx

```
__float128 Color128::Zx
```

6.3.1.10 Zx2

```
__float128 Color128::Zx2
```

6.3.1.11 Zy

```
__float128 Color128::Zy
```

6.3.1.12 Zy2

```
__float128 Color128::Zy2
```

The documentation for this struct was generated from the following file:

- colors.h

6.4 ColorDBL Struct Reference

Data Fields

- double **Zx**
- double **Zy**
- double **Cx**
- double **Cy**
- double **Zx2**
- double **Zy2**
- double complex **Z**
- double complex **C**
- double **colorPoly**
- double **Exps**
- double **DistanceMax**
- int **n**

6.4.1 Field Documentation

6.4.1.1 C

```
double complex ColorDBL::C
```

6.4.1.2 colorPoly

```
double ColorDBL::colorPoly
```

6.4.1.3 Cx

```
double ColorDBL::Cx
```

6.4.1.4 Cy

```
double ColorDBL::Cy
```

6.4.1.5 DistanceMax

```
double ColorDBL::DistanceMax
```

6.4.1.6 Exps

```
double ColorDBL::Exps
```

6.4.1.7 n

```
int ColorDBL::n
```

6.4.1.8 Z

```
double complex ColorDBL::Z
```

6.4.1.9 Zx

```
double ColorDBL::Zx
```

6.4.1.10 Zx2

```
double ColorDBL::Zx2
```

6.4.1.11 Zy

```
double ColorDBL::Zy
```

6.4.1.12 Zy2

```
double ColorDBL::Zy2
```

The documentation for this struct was generated from the following file:

- colors.h

6.5 ColorFLT Struct Reference

Data Fields

- float **Zx**
- float **Zy**
- float **Cx**
- float **Cy**
- float **Zx2**
- float **Zy2**
- float **colorPoly**
- float **Exps**
- float **DistanceMax**
- complex **C**
- complex **Z**
- int **n**

6.5.1 Field Documentation

6.5.1.1 C

```
complex ColorFLT::C
```

6.5.1.2 colorPoly

```
float ColorFLT::colorPoly
```

6.5.1.3 Cx

```
float ColorFLT::Cx
```

6.5.1.4 Cy

```
float ColorFLT::Cy
```

6.5.1.5 DistanceMax

```
float ColorFLT::DistanceMax
```

6.5.1.6 Exps

```
float ColorFLT::Exps
```

6.5.1.7 n

```
int ColorFLT::n
```

6.5.1.8 Z

```
complex ColorFLT::Z
```

6.5.1.9 Zx

```
float ColorFLT::Zx
```

6.5.1.10 Zx2

```
float ColorFLT::Zx2
```

6.5.1.11 Zy

```
float ColorFLT::Zy
```

6.5.1.12 Zy2

```
float ColorFLT::Zy2
```

The documentation for this struct was generated from the following file:

- colors.h

6.6 ColorInfo Struct Reference

Data Fields

- const char * **name**
- const char * **hex**
- Rgb **rgb**

6.6.1 Field Documentation

6.6.1.1 hex

```
const char* ColorInfo::hex
```

6.6.1.2 name

```
const char* ColorInfo::name
```

6.6.1.3 rgb

```
Rgb ColorInfo::rgb
```

The documentation for this struct was generated from the following file:

- cnames.h

6.7 ColorLDBL Struct Reference

Data Fields

- long double **Zx**
- long double **Zy**
- long double **Cx**
- long double **Cy**
- long double **Zx2**
- long double **Zy2**
- long double complex **Z**
- long double complex **C**
- long double **colorPoly**
- long double **Exps**
- long double **DistanceMax**
- int **n**

6.7.1 Field Documentation

6.7.1.1 C

```
long double complex ColorLDBL::C
```

6.7.1.2 colorPoly

```
long double ColorLDBL::colorPoly
```

6.7.1.3 Cx

```
long double ColorLDBL::Cx
```

6.7.1.4 Cy

```
long double ColorLDBL::Cy
```

6.7.1.5 DistanceMax

```
long double ColorLDBL::DistanceMax
```

6.7.1.6 Exps

```
long double ColorLDBL::Exps
```

6.7.1.7 n

```
int ColorLDBL::n
```

6.7.1.8 Z

```
long double complex ColorLDBL::Z
```

6.7.1.9 Zx

```
long double ColorLDBL::Zx
```

6.7.1.10 Zx2

```
long double ColorLDBL::Zx2
```

6.7.1.11 Zy

```
long double ColorLDBL::Zy
```

6.7.1.12 Zy2

```
long double ColorLDBL::Zy2
```

The documentation for this struct was generated from the following file:

- colors.h

6.8 error Struct Reference

Data Fields

- const unsigned char * **json**
- size_t **position**

6.8.1 Field Documentation

6.8.1.1 json

```
const unsigned char* error::json
```

6.8.1.2 position

```
size_t error::position
```

The documentation for this struct was generated from the following file:

- cJSON.c

6.9 HSV Struct Reference

Data Fields

- double **H**
- double **S**
- double **V**

6.9.1 Field Documentation

6.9.1.1 H

```
double HSV::H
```

6.9.1.2 S

```
double HSV::S
```

6.9.1.3 V

```
double HSV::V
```

The documentation for this struct was generated from the following file:

- cspace.h

6.10 internal_hooks Struct Reference

Data Fields

- void ∗(∗ **allocate**)(size_t size)
- void(∗ **deallocate**)(void ∗pointer)
- void ∗(∗ **reallocate**)(void ∗pointer, size_t size)

The documentation for this struct was generated from the following file:

- cJSON.c

6.11 parse_buffer Struct Reference

Data Fields

- const unsigned char ∗ **content**
- size_t **length**
- size_t **offset**
- size_t **depth**
- internal_hooks **hooks**

6.11.1 Field Documentation

6.11.1.1 content

```
const unsigned char* parse_buffer::content
```

6.11.1.2 depth

```
size_t parse_buffer::depth
```

6.11.1.3 hooks

```
internal_hooks parse_buffer::hooks
```

6.11.1.4 length

```
size_t parse_buffer::length
```

6.11.1.5 offset

```
size_t parse_buffer::offset
```

The documentation for this struct was generated from the following file:

- cJSON.c

6.12 printbuffer Struct Reference

Data Fields

- unsigned char * **buffer**
- size_t **length**
- size_t **offset**
- size_t **depth**
- cJSON_bool **noalloc**
- cJSON_bool **format**
- internal_hooks **hooks**

6.12.1 Field Documentation

6.12.1.1 buffer

```
unsigned char* printbuffer::buffer
```

6.12.1.2 depth

```
size_t printbuffer::depth
```

6.12.1.3 format

```
cJSON_bool printbuffer::format
```

6.12.1.4 hooks

```
internal_hooks printbuffer::hooks
```

6.12.1.5 length

```
size_t printbuffer::length
```

6.12.1.6 noalloc

```
cJSON_bool printbuffer::noalloc
```

6.12.1.7 offset

```
size_t printbuffer::offset
```

The documentation for this struct was generated from the following file:

- cJSON.c

Chapter 7

File Documentation

7.1 cnames.c File Reference

A collection of functions in aid of Rgb naming and value retrieval.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "palette.h"
#include "cnames.h"
#include "getopt.h"
```

Macros

- #define **__USE_MINGW_ANSI_STDIO** 1

Functions

- bool str2Rgb (char *s, Rgb *color)

  Convert named Rgb to values.
- bool RgbNE (Rgb a, Rgb b)

  NE comparison for Rgb pair.
- bool RgbEQ (Rgb a, Rgb b)

  EQ comparison for Rgb pair.
- char * RgbName (Rgb color)

  Find name of Rgb value.

Variables

- const ColorInfo **color_data** [COLOR_NAMES_MAX]

7.1.1 Detailed Description

A collection of functions in aid of Rgb naming and value retrieval.

Author

Hugh S. Myers

Date

Fri Aug 11 06:49:25 2017

7.1.2 DESCRIPTION

The functions in this file allow fetching the names and values of a particular Rgb instance. As collateral damage this provides tests for equality and its inverse.

Note

gcc -Wall -DCNAMES_TEST -o nam cnames.c gcc -ggdb -Wall -DCNAMES_TEST -o nam cnames.c

Bug No known bugs.

7.1.3 Function Documentation

7.1.3.1 RgbEQ()

```
bool RgbEQ (
            Rgb a,
            Rgb b )
```

EQ comparison for Rgb pair.

This routine does a member comparison for the equality of two Rgb values.

Parameters

a	First Rgb value.
b	Second Rgb value.

Returns

Returns true for equality, false otherwise.

7.1.3.2 RgbName()

```
char* RgbName (
            Rgb color )
```

Find name of Rgb value.

Simple linear look-up seeking the text name of a given Rgb value.

Parameters

color	Rgb value to seek.

Returns

Character pointer upon success, NULL otherwise.

7.1.3.3 RgbNE()

```
bool RgbNE (
            Rgb a,
            Rgb b )
```

NE comparison for Rgb pair.

This routine does a member comparison for the non-equality of two Rgb values.

Parameters

a	First Rgb value.
b	Second Rgb value.

Returns

Returns true for non-equality, false otherwise.

7.1.3.4 str2Rgb()

```
bool str2Rgb (
            char * s,
            Rgb * color )
```

Convert named Rgb to values.

This routine does a simple linear look-up through the color_data table and transfers the RGB values upon success.

Parameters

s	Color name.
color	Target Rgb.

Returns

True for success, false for failure.

7.2 dictionary.c File Reference

Implements a dictionary for string variables.

```
#include "dictionary.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

Macros

- #define MAXVALSZ 1024
- #define DICTMINSZ 128
- #define DICT_INVALID_KEY ((char*)-1)

Functions

- unsigned dictionary_hash (const char *key)

  Compute the hash key for a string.
- dictionary * dictionary_new (size_t size)

  Create a new dictionary object.
- void dictionary_del (dictionary *d)

  Delete a dictionary object.
- const char * dictionary_get (const dictionary *d, const char *key, const char *def)

  Get a value from a dictionary.
- int dictionary_set (dictionary *d, const char *key, const char *val)

  Set a value in a dictionary.
- void dictionary_unset (dictionary *d, const char *key)

  Delete a key in a dictionary.
- void dictionary_dump (const dictionary *d, FILE *out)

  Dump a dictionary to an opened file pointer.

7.2.1 Detailed Description

Implements a dictionary for string variables.

Author

N. Devillard This module implements a simple dictionary object, i.e. a list of string/string associations. This object is useful to store e.g. informations retrieved from a configuration file (ini files).

7.2.2 Macro Definition Documentation

7.2.2.1 DICT_INVALID_KEY

```
#define DICT_INVALID_KEY ((char*)-1)
```

Invalid key token

7.2.2.2 DICTMINSZ

```
#define DICTMINSZ 128
```

Minimal allocated number of entries in a dictionary

7.2.2.3 MAXVALSZ

```
#define MAXVALSZ 1024
```

Maximum value size for integers and doubles.

7.2.3 Function Documentation

7.2.3.1 dictionary_del()

```
void dictionary_del (
            dictionary * d )
```

Delete a dictionary object.

Parameters

d	dictionary object to deallocate.

Returns

void

Deallocate a dictionary object and all memory associated to it.

7.2.3.2 dictionary_dump()

```
void dictionary_dump (
            const dictionary * d,
            FILE * out )
```

Dump a dictionary to an opened file pointer.

Parameters

d	Dictionary to dump
f	Opened file pointer.

Returns

void

Dumps a dictionary onto an opened file pointer. Key pairs are printed out as [Key]=[Value], one per line. It is Ok to provide stdout or stderr as output file pointers.

7.2.3.3 dictionary_get()

```
const char* dictionary_get (
            const dictionary * d,
            const char * key,
            const char * def )
```

Get a value from a dictionary.

Parameters

d	dictionary object to search.
key	Key to look for in the dictionary.
def	Default value to return if key not found.

Returns

1 pointer to internally allocated character string.

This function locates a key in a dictionary and returns a pointer to its value, or the passed 'def' pointer if no such key can be found in dictionary. The returned character pointer points to data internal to the dictionary object, you should not try to free it or modify it.

7.2.3.4 dictionary_hash()

```
unsigned dictionary_hash (
            const char * key )
```

Compute the hash key for a string.

Parameters

key	Character string to use for key.

Returns

1 unsigned int on at least 32 bits.

This hash function has been taken from an Article in Dr Dobbs Journal. This is normally a collision-free function, distributing keys evenly. The key is stored anyway in the struct so that collision can be avoided by comparing the key itself in last resort.

7.2.3.5 dictionary_new()

```
dictionary* dictionary_new (
            size_t size )
```

Create a new dictionary object.

Parameters

size	Optional initial size of the dictionary.

Returns

1 newly allocated dictionary objet.

This function allocates a new dictionary object of given size and returns it. If you do not know in advance (roughly) the number of entries in the dictionary, give size=0.

7.2.3.6 dictionary_set()

```
int dictionary_set (
            dictionary * d,
            const char * key,
            const char * val )
```

Set a value in a dictionary.

Parameters

d	dictionary object to modify.
key	Key to modify or add.
val	Value to add.

Returns

int 0 if Ok, anything else otherwise

If the given key is found in the dictionary, the associated value is replaced by the provided one. If the key cannot be found in the dictionary, it is added to it.

It is Ok to provide a NULL value for val, but NULL values for the dictionary or the key are considered as errors: the function will return immediately in such a case.

Notice that if you dictionary_set a variable to NULL, a call to dictionary_get will return a NULL value: the variable will be found, and its value (NULL) is returned. In other words, setting the variable content to NULL is equivalent to deleting the variable from the dictionary. It is not possible (in this implementation) to have a key in the dictionary without value.

This function returns non-zero in case of failure.

7.2.3.7 dictionary_unset()

```
void dictionary_unset (
            dictionary * d,
            const char * key )
```

Delete a key in a dictionary.

Parameters

d	dictionary object to modify.
key	Key to remove.

Returns

void

This function deletes a key in a dictionary. Nothing is done if the key cannot be found.

7.3 dictionary.h File Reference

Implements a dictionary for string variables.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

Data Structures

- struct dictionary

  Dictionary object. *More...*

Functions

- unsigned dictionary_hash (const char *key)

  Compute the hash key for a string.
- dictionary * dictionary_new (size_t size)

  Create a new dictionary object.
- void dictionary_del (dictionary *vd)

  Delete a dictionary object.
- const char * dictionary_get (const dictionary *d, const char *key, const char *def)

  Get a value from a dictionary.
- int dictionary_set (dictionary *vd, const char *key, const char *val)

  Set a value in a dictionary.
- void dictionary_unset (dictionary *d, const char *key)

  Delete a key in a dictionary.
- void dictionary_dump (const dictionary *d, FILE *out)

  Dump a dictionary to an opened file pointer.

7.3.1 Detailed Description

Implements a dictionary for string variables.

Author

N. Devillard This module implements a simple dictionary object, i.e. a list of string/string associations. This object is useful to store e.g. informations retrieved from a configuration file (ini files).

7.3.2 Data Structure Documentation

7.3.2.1 struct dictionary

Dictionary object.

This object contains a list of string/string associations. Each association is identified by a unique string key. Looking up values in the dictionary is speeded up by the use of a (hopefully collision-free) hash function.

Data Fields

unsigned *	hash	List of string keys
char **	key	List of string values
int	n	
ssize_t	size	Number of entries in dictionary
char **	val	Storage size

7.3.3 Function Documentation

7.3.3.1 dictionary_del()

```
void dictionary_del (
            dictionary * d )
```

Delete a dictionary object.

Parameters

d	dictionary object to deallocate.

Returns

void

Deallocate a dictionary object and all memory associated to it.

7.3.3.2 dictionary_dump()

```
void dictionary_dump (
            const dictionary * d,
            FILE * out )
```

Dump a dictionary to an opened file pointer.

Parameters

d	Dictionary to dump
f	Opened file pointer.

Returns

void

Dumps a dictionary onto an opened file pointer. Key pairs are printed out as [Key]=[Value], one per line. It is Ok to provide stdout or stderr as output file pointers.

7.3.3.3 dictionary_get()

```
const char* dictionary_get (
            const dictionary * d,
            const char * key,
            const char * def )
```

Get a value from a dictionary.

Parameters

d	dictionary object to search.
key	Key to look for in the dictionary.
def	Default value to return if key not found.

Returns

1 pointer to internally allocated character string.

This function locates a key in a dictionary and returns a pointer to its value, or the passed 'def' pointer if no such key can be found in dictionary. The returned character pointer points to data internal to the dictionary object, you should not try to free it or modify it.

7.3.3.4 dictionary_hash()

```
unsigned dictionary_hash (
            const char * key )
```

Compute the hash key for a string.

Parameters

key	Character string to use for key.

Returns

1 unsigned int on at least 32 bits.

This hash function has been taken from an Article in Dr Dobbs Journal. This is normally a collision-free function, distributing keys evenly. The key is stored anyway in the struct so that collision can be avoided by comparing the key itself in last resort.

7.3.3.5 dictionary_new()

```
dictionary* dictionary_new (
            size_t size )
```

Create a new dictionary object.

Parameters

size	Optional initial size of the dictionary.

Returns

1 newly allocated dictionary objet.

This function allocates a new dictionary object of given size and returns it. If you do not know in advance (roughly) the number of entries in the dictionary, give size=0.

7.3.3.6 dictionary_set()

```
int dictionary_set (
            dictionary * d,
```

```
            const char * key,
            const char * val )
```

Set a value in a dictionary.

Parameters

d	dictionary object to modify.
key	Key to modify or add.
val	Value to add.

Returns

int 0 if Ok, anything else otherwise

If the given key is found in the dictionary, the associated value is replaced by the provided one. If the key cannot be found in the dictionary, it is added to it.

It is Ok to provide a NULL value for val, but NULL values for the dictionary or the key are considered as errors: the function will return immediately in such a case.

Notice that if you dictionary_set a variable to NULL, a call to dictionary_get will return a NULL value: the variable will be found, and its value (NULL) is returned. In other words, setting the variable content to NULL is equivalent to deleting the variable from the dictionary. It is not possible (in this implementation) to have a key in the dictionary without value.

This function returns non-zero in case of failure.

7.3.3.7 dictionary_unset()

```
void dictionary_unset (
            dictionary * d,
            const char * key )
```

Delete a key in a dictionary.

Parameters

d	dictionary object to modify.
key	Key to remove.

Returns

void

This function deletes a key in a dictionary. Nothing is done if the key cannot be found.

7.4 elapsed.c File Reference

Routines to provide time information as mp executes.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>
#include <string.h>
#include "elapsed.h"
#include "util.h"
```

Functions

- void **elapsed** (int elapsedSeconds, long milleseconds)
- int **timeval_subtract** (struct timeval *result, struct timeval *x, struct timeval *y)
- void **timeval_print** (struct timeval *tv)
- ssize_t **format_timeval** (struct timeval *tv, char *buf, size_t sz)

7.4.1 Detailed Description

Routines to provide time information as mp executes.

Author

Hugh S. Myers

Date

Mon Jul 24 16:09:35 2017

Description

Bug No known bugs.

7.5 getopt.c File Reference

Implementation of getopt_long for mp.

```
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <string.h>
#include <sys/types.h>
#include <dirent.h>
#include <ctype.h>
#include <quadmath.h>
#include "iniparser.h"
#include "cJSON.h"
#include "util.h"
#include "cnames.h"
```

Macros

- #define **__USE_MINGW_ANSI_STDIO** 1

Functions

- Parameters **getParameters** (int argc, char ∗argv[ ], char ∗Version, char ∗Date)
- Parameters **fromJSON** (Parameters p)
- void **toJSON** (Parameters g)
- Parameters **rawCl** (int argc, char ∗argv[ ], Parameters p)
- void **help** (char c, char ∗Program, char ∗Version, char ∗Date)
- Parameters **parse_ini_file** (char ∗ini_name, Parameters g)
- int **getNext** (char ∗path)
- char ∗ **getstr** (char ∗arg)
- void **showParameters** (Parameters p, char ∗s)
- Parameters **zeroP** (void)

7.5.1 Detailed Description

Implementation of getopt_long for mp.

Author

Hugh S. Myers

Date

Mon Jul 24 15:05:30 2017

These functions allow mp to garner information from a variety of sources. From the command line, both with and without the traditional dashes, as well as from an 'ini' or configuration file. Sufficient information can be gathered from .JSON files as well,

Note

gcc -Wall -DGETOPT_TEST cnames.c cJSON.c getopt.c dictionary.c iniparser.c util.c -o getopt -lquadmath
gcc -ggdb -Wall -DGETOPT_TEST cnames.c cJSON.c getopt.c dictionary.c iniparser.c util.c -o getopt -lquadmath

Bug No known bugs.

7.6 getopt.h File Reference

A collection of functions in aid of creating a parameter object to carry around.

```
#include <stdbool.h>
#include <complex.h>
#include <quadmath.h>
```

Data Structures

- struct Parameters

Macros

- #define **NAMES_SIZE** 16

Enumerations

- enum **innerLoop** { **COMPLEX**, **ZX**, **DEMM**, **MSETPOT** }

Functions

- char * **getstr** (char *arg)
- int **getNext** (char *path)
- Parameters **fromJSON** (Parameters p)
- Parameters **getParameters** (int argc, char *argv[ ], char *Version, char *Date)
- Parameters **rawCl** (int argc, char *argv[ ], Parameters p)
- Parameters **parse_ini_file** (char *ini_name, Parameters p)
- Parameters **zeroP** (void)
- void **help** (char c, char *Program, char *Version, char *Date)
- void **showParameters** (Parameters p, char *s)
- void **toJSON** (Parameters g)

7.6.1 Detailed Description

A collection of functions in aid of creating a parameter object to carry around.

Author

Hugh S. Myers

Date

Tue Jul 25 11:49:07 2017

7.6.2 DESCRIPTION

Function declarations. More importantly the typedef for Parameters.

Bug No known bugs.

7.6.3 Data Structure Documentation

7.6.3.1 struct Parameters

Data Fields

char *	aa	
int	bourke	
__complex128	center	
__float128	centerX	
__float128	centerY	
int	color	
char *	config	
int	cv	
__float128	diameter	
float	escape	
char *	filename	
int	height	
__float128	height2	
char *	json	
__float128	magnify	
int	maxiter	
char *	names[NAMES_SIZE]	
int	nargc	
int	next	
int	nMax	
int	old	
char *	palname	
__float128	radius	
int	tweak	
int	width	
__float128	width2	
int	writeJSON	

7.7 iniparser.c File Reference

Parser for ini files.

```
#include <ctype.h>
#include <stdarg.h>
#include "iniparser.h"
```

Macros

- #define **__USE_MINGW_ANSI_STDIO** 1
- #define **ASCIILINESZ** (1024)
- #define **INI_INVALID_KEY** ((char*)-1)

Enumerations

- enum line_status {
  LINE_UNPROCESSED, **LINE_ERROR**, **LINE_EMPTY**, **LINE_COMMENT**,
  LINE_SECTION, **LINE_VALUE** }

Functions

- int iniparser_getnsec (const dictionary *d)

  Get number of sections in a dictionary.
- const char * iniparser_getsecname (const dictionary *d, int n)

  Get name for section n in a dictionary.
- void iniparser_dump (const dictionary *d, FILE *f)

  Dump a dictionary to an opened file pointer.
- void iniparser_dump_ini (const dictionary *d, FILE *f)

  Save a dictionary to a loadable ini file.
- void iniparser_dumpsection_ini (const dictionary *d, const char *s, FILE *f)

  Save a dictionary section to a loadable ini file.
- int iniparser_getsecnkeys (const dictionary *d, const char *s)

  Get the number of keys in a section of a dictionary.
- const char ** iniparser_getseckeys (const dictionary *d, const char *s, const char **keys)

  Get the number of keys in a section of a dictionary.
- const char * iniparser_getstring (const dictionary *d, const char *key, const char *def)

  Get the string associated to a key.
- long int iniparser_getlongint (const dictionary *d, const char *key, long int notfound)

  Get the string associated to a key, convert to an long int.
- int iniparser_getint (const dictionary *d, const char *key, int notfound)

  Get the string associated to a key, convert to an int.
- double iniparser_getdouble (const dictionary *d, const char *key, double notfound)

  Get the string associated to a key, convert to a double.
- long double iniparser_getlongdouble (const dictionary *d, const char *key, long double notfound)

  Get the string associated to a key, convert to a long double.
- int iniparser_getboolean (const dictionary *d, const char *key, int notfound)

  Get the string associated to a key, convert to a boolean.
- int iniparser_find_entry (const dictionary *ini, const char *entry)

  Finds out if a given entry exists in a dictionary.
- int iniparser_set (dictionary *ini, const char *entry, const char *val)

  Set an entry in a dictionary.
- void iniparser_unset (dictionary *ini, const char *entry)

  Delete an entry in a dictionary.
- dictionary * iniparser_load (const char *ininame)

  Parse an ini file and return an allocated dictionary object.
- void iniparser_freedict (dictionary *d)

  Free all memory associated to an ini dictionary.

7.7.1 Detailed Description

Parser for ini files.

Author

N. Devillard

7.7.2 Enumeration Type Documentation

7.7.2.1 line_status

```
enum line_status
```

This enum stores the status for each parsed line (internal use only).

7.7.3 Function Documentation

7.7.3.1 iniparser_dump()

```
void iniparser_dump (
            const dictionary * d,
            FILE * f )
```

Dump a dictionary to an opened file pointer.

Parameters

d	Dictionary to dump.
f	Opened file pointer to dump to.

Returns

void

This function prints out the contents of a dictionary, one element by line, onto the provided file pointer. It is OK to specify `stderr` or `stdout` as output files. This function is meant for debugging purposes mostly.

7.7.3.2 iniparser_dump_ini()

```
void iniparser_dump_ini (
            const dictionary * d,
            FILE * f )
```

Save a dictionary to a loadable ini file.

Parameters

d	Dictionary to dump
f	Opened file pointer to dump to

Returns

void

This function dumps a given dictionary into a loadable ini file. It is Ok to specify `stderr` or `stdout` as output files.

7.7.3.3 iniparser_dumpsection_ini()

```
void iniparser_dumpsection_ini (
            const dictionary * d,
            const char * s,
            FILE * f )
```

Save a dictionary section to a loadable ini file.

Parameters

d	Dictionary to dump
s	Section name of dictionary to dump
f	Opened file pointer to dump to

Returns

void

This function dumps a given section of a given dictionary into a loadable ini file. It is Ok to specify `stderr` or `stdout` as output files.

7.7.3.4 iniparser_find_entry()

```
int iniparser_find_entry (
            const dictionary * ini,
            const char * entry )
```

Finds out if a given entry exists in a dictionary.

Parameters

ini	Dictionary to search
entry	Name of the entry to look for

Returns

integer 1 if entry exists, 0 otherwise

Finds out if a given entry exists in the dictionary. Since sections are stored as keys with NULL associated values, this is the only way of querying for the presence of sections in a dictionary.

7.7.3.5 iniparser_freedict()

```
void iniparser_freedict (
            dictionary * d )
```

Free all memory associated to an ini dictionary.

Parameters

d	Dictionary to free

Returns

void

Free all memory associated to an ini dictionary. It is mandatory to call this function before the dictionary object gets out of the current context.

7.7.3.6 iniparser_getboolean()

```
int iniparser_getboolean (
            const dictionary * d,
            const char * key,
            int notfound )
```

Get the string associated to a key, convert to a boolean.

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

integer

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

A true boolean is found if one of the following is matched:

- A string starting with 'y'
- A string starting with 'Y'
- A string starting with 't'
- A string starting with 'T'
- A string starting with '1'

A false boolean is found if one of the following is matched:

- A string starting with 'n'
- A string starting with 'N'
- A string starting with 'f'
- A string starting with 'F'
- A string starting with '0'

The notfound value returned if no boolean is identified, does not necessarily have to be 0 or 1.

7.7.3.7 iniparser_getdouble()

```
double iniparser_getdouble (
            const dictionary * d,
            const char * key,
            double notfound )
```

Get the string associated to a key, convert to a double.

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

double

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

7.7.3.8 iniparser_getint()

```
int iniparser_getint (
            const dictionary * d,
            const char * key,
            int notfound )
```

Get the string associated to a key, convert to an int.

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

integer

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

Supported values for integers include the usual C notation so decimal, octal (starting with 0) and hexadecimal (starting with 0x) are supported. Examples:

"42" -> 42 "042" -> 34 (octal -> decimal) "0x42" -> 66 (hexa -> decimal)

Warning: the conversion may overflow in various ways. Conversion is totally outsourced to strtol(), see the associated man page for overflow handling.

Credits: Thanks to A. Becker for suggesting strtol()

7.7.3.9 iniparser_getlongdouble()

```
long double iniparser_getlongdouble (
            const dictionary * d,
            const char * key,
            long double notfound )
```

Get the string associated to a key, convert to a long double.

Get the string associated to a key, convert to a double.

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

long double

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

7.7.3.10 iniparser_getlongint()

```
long int iniparser_getlongint (
            const dictionary * d,
            const char * key,
            long int notfound )
```

Get the string associated to a key, convert to an long int.

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

long integer

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

Supported values for integers include the usual C notation so decimal, octal (starting with 0) and hexadecimal (starting with 0x) are supported. Examples:

"42" -> 42 "042" -> 34 (octal -> decimal) "0x42" -> 66 (hexa -> decimal)

Warning: the conversion may overflow in various ways. Conversion is totally outsourced to strtol(), see the associated man page for overflow handling.

Credits: Thanks to A. Becker for suggesting strtol()

7.7.3.11 iniparser_getnsec()

```
int iniparser_getnsec (
            const dictionary * d )
```

Get number of sections in a dictionary.

Parameters

d	Dictionary to examine

Returns

int Number of sections found in dictionary

This function returns the number of sections found in a dictionary. The test to recognize sections is done on the string stored in the dictionary: a section name is given as "section" whereas a key is stored as "section:key", thus the test looks for entries that do not contain a colon.

This clearly fails in the case a section name contains a colon, but this should simply be avoided.

This function returns -1 in case of error.

7.7.3.12 iniparser_getseckeys()

```
const char** iniparser_getseckeys (
            const dictionary * d,
            const char * s,
            const char ** keys )
```

Get the number of keys in a section of a dictionary.

Parameters

d	Dictionary to examine
s	Section name of dictionary to examine
keys	Already allocated array to store the keys in

Returns

The pointer passed as `keys` argument or NULL in case of error

This function queries a dictionary and finds all keys in a given section. The keys argument should be an array of pointers which size has been determined by calling `iniparser_getsecnkeys` function prior to this one.

Each pointer in the returned char pointer-to-pointer is pointing to a string allocated in the dictionary; do not free or modify them.

7.7.3.13 iniparser_getsecname()

```
const char* iniparser_getsecname (
            const dictionary * d,
            int n )
```

Get name for section n in a dictionary.

Parameters

d	Dictionary to examine
n	Section number (from 0 to nsec-1).

Returns

Pointer to char string

This function locates the n-th section in a dictionary and returns its name as a pointer to a string statically allocated inside the dictionary. Do not free or modify the returned string!

This function returns NULL in case of error.

7.7.3.14 iniparser_getsecnkeys()

```
int iniparser_getsecnkeys (
            const dictionary * d,
            const char * s )
```

Get the number of keys in a section of a dictionary.

Parameters

d	Dictionary to examine
s	Section name of dictionary to examine

Returns

Number of keys in section

7.7.3.15 iniparser_getstring()

```
const char* iniparser_getstring (
            const dictionary * d,
            const char * key,
            const char * def )
```

Get the string associated to a key.

Parameters

d	Dictionary to search
key	Key string to look for
def	Default value to return if key not found.

Returns

pointer to statically allocated character string

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the pointer passed as 'def' is returned. The returned char pointer is pointing to a string allocated in the dictionary, do not free or modify it.

7.7.3.16 iniparser_load()

```
dictionary* iniparser_load (
            const char * ininame )
```

Parse an ini file and return an allocated dictionary object.

Parameters

ininame	Name of the ini file to read.

Returns

Pointer to newly allocated dictionary

This is the parser for ini files. This function is called, providing the name of the file to be read. It returns a dictionary object that should not be accessed directly, but through accessor functions instead.

The returned dictionary must be freed using iniparser_freedict().

7.7.3.17 iniparser_set()

```
int iniparser_set (
            dictionary * ini,
            const char * entry,
            const char * val )
```

Set an entry in a dictionary.

Parameters

ini	Dictionary to modify.
entry	Entry to modify (entry name)
val	New value to associate to the entry.

Returns

int 0 if Ok, -1 otherwise.

If the given entry can be found in the dictionary, it is modified to contain the provided value. If it cannot be found, the entry is created. It is Ok to set val to NULL.

7.7.3.18 iniparser_unset()

```
void iniparser_unset (
            dictionary * ini,
            const char * entry )
```

Delete an entry in a dictionary.

Parameters

ini	Dictionary to modify
entry	Entry to delete (entry name)

Returns

void

If the given entry can be found, it is deleted from the dictionary.

7.8 iniparser.h File Reference

Parser for ini files.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "dictionary.h"
```

Functions

- void iniparser_set_error_callback (int(∗errback)(const char ∗,...))

  Configure a function to receive the error messages.
- int iniparser_getnsec (const dictionary ∗d)

  Get number of sections in a dictionary.
- const char ∗ iniparser_getsecname (const dictionary ∗d, int n)

  Get name for section n in a dictionary.
- void iniparser_dump_ini (const dictionary ∗d, FILE ∗f)

  Save a dictionary to a loadable ini file.
- void iniparser_dumpsection_ini (const dictionary ∗d, const char ∗s, FILE ∗f)

  Save a dictionary section to a loadable ini file.

- void iniparser_dump (const dictionary *d, FILE *f)
  Dump a dictionary to an opened file pointer.
- int iniparser_getsecnkeys (const dictionary *d, const char *s)
  Get the number of keys in a section of a dictionary.
- const char ** iniparser_getseckeys (const dictionary *d, const char *s, const char **keys)
  Get the number of keys in a section of a dictionary.
- const char * iniparser_getstring (const dictionary *d, const char *key, const char *def)
  Get the string associated to a key.
- int iniparser_getint (const dictionary *d, const char *key, int notfound)
  Get the string associated to a key, convert to an int.
- long int iniparser_getlongint (const dictionary *d, const char *key, long int notfound)
  Get the string associated to a key, convert to an long int.
- double iniparser_getdouble (const dictionary *d, const char *key, double notfound)
  Get the string associated to a key, convert to a double.
- long double iniparser_getlongdouble (const dictionary *d, const char *key, long double notfound)
  Get the string associated to a key, convert to a double.
- int iniparser_getboolean (const dictionary *d, const char *key, int notfound)
  Get the string associated to a key, convert to a boolean.
- int iniparser_set (dictionary *ini, const char *entry, const char *val)
  Set an entry in a dictionary.
- void iniparser_unset (dictionary *ini, const char *entry)
  Delete an entry in a dictionary.
- int iniparser_find_entry (const dictionary *ini, const char *entry)
  Finds out if a given entry exists in a dictionary.
- dictionary * iniparser_load (const char *ininame)
  Parse an ini file and return an allocated dictionary object.
- void iniparser_freedict (dictionary *d)
  Free all memory associated to an ini dictionary.

7.8.1 Detailed Description

Parser for ini files.

Author

N. Devillard

7.8.2 Function Documentation

7.8.2.1 iniparser_dump()

```
void iniparser_dump (
            const dictionary * d,
            FILE * f )
```

Dump a dictionary to an opened file pointer.

Parameters

d	Dictionary to dump.
f	Opened file pointer to dump to.

Returns

void

This function prints out the contents of a dictionary, one element by line, onto the provided file pointer. It is OK to specify `stderr` or `stdout` as output files. This function is meant for debugging purposes mostly.

7.8.2.2 iniparser_dump_ini()

```
void iniparser_dump_ini (
              const dictionary * d,
              FILE * f )
```

Save a dictionary to a loadable ini file.

Parameters

d	Dictionary to dump
f	Opened file pointer to dump to

Returns

void

This function dumps a given dictionary into a loadable ini file. It is Ok to specify `stderr` or `stdout` as output files.

7.8.2.3 iniparser_dumpsection_ini()

```
void iniparser_dumpsection_ini (
              const dictionary * d,
              const char * s,
              FILE * f )
```

Save a dictionary section to a loadable ini file.

Parameters

d	Dictionary to dump
s	Section name of dictionary to dump
f	Opened file pointer to dump to

Returns

void

This function dumps a given section of a given dictionary into a loadable ini file. It is Ok to specify `stderr` or `stdout` as output files.

7.8.2.4 iniparser_find_entry()

```
int iniparser_find_entry (
            const dictionary * ini,
            const char * entry )
```

Finds out if a given entry exists in a dictionary.

Parameters

ini	Dictionary to search
entry	Name of the entry to look for

Returns

integer 1 if entry exists, 0 otherwise

Finds out if a given entry exists in the dictionary. Since sections are stored as keys with NULL associated values, this is the only way of querying for the presence of sections in a dictionary.

7.8.2.5 iniparser_freedict()

```
void iniparser_freedict (
            dictionary * d )
```

Free all memory associated to an ini dictionary.

Parameters

d	Dictionary to free

Returns

void

Free all memory associated to an ini dictionary. It is mandatory to call this function before the dictionary object gets out of the current context.

7.8.2.6 iniparser_getboolean()

```
int iniparser_getboolean (
            const dictionary * d,
```

```
            const char * key,
            int notfound )
```

Get the string associated to a key, convert to a boolean.

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

integer

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

A true boolean is found if one of the following is matched:

- A string starting with 'y'
- A string starting with 'Y'
- A string starting with 't'
- A string starting with 'T'
- A string starting with '1'

A false boolean is found if one of the following is matched:

- A string starting with 'n'
- A string starting with 'N'
- A string starting with 'f'
- A string starting with 'F'
- A string starting with '0'

The notfound value returned if no boolean is identified, does not necessarily have to be 0 or 1.

7.8.2.7 iniparser_getdouble()

```
double iniparser_getdouble (
            const dictionary * d,
            const char * key,
            double notfound )
```

Get the string associated to a key, convert to a double.

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

double

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

7.8.2.8 iniparser_getint()

```
int iniparser_getint (
            const dictionary * d,
            const char * key,
            int notfound )
```

Get the string associated to a key, convert to an int.

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

integer

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

Supported values for integers include the usual C notation so decimal, octal (starting with 0) and hexadecimal (starting with 0x) are supported. Examples:

- "42" -> 42
- "042" -> 34 (octal -> decimal)
- "0x42" -> 66 (hexa -> decimal)

Warning: the conversion may overflow in various ways. Conversion is totally outsourced to strtol(), see the associated man page for overflow handling.

Credits: Thanks to A. Becker for suggesting strtol()

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

integer

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

Supported values for integers include the usual C notation so decimal, octal (starting with 0) and hexadecimal (starting with 0x) are supported. Examples:

"42" -> 42 "042" -> 34 (octal -> decimal) "0x42" -> 66 (hexa -> decimal)

Warning: the conversion may overflow in various ways. Conversion is totally outsourced to strtol(), see the associated man page for overflow handling.

Credits: Thanks to A. Becker for suggesting strtol()

7.8.2.9 iniparser_getlongdouble()

```
long double iniparser_getlongdouble (
             const dictionary * d,
             const char * key,
             long double notfound )
```

Get the string associated to a key, convert to a double.

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

long double

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

Get the string associated to a key, convert to a double.

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

long double

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

7.8.2.10 iniparser_getlongint()

```
long int iniparser_getlongint (
            const dictionary * d,
            const char * key,
            long int notfound )
```

Get the string associated to a key, convert to an long int.

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

integer

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

Supported values for integers include the usual C notation so decimal, octal (starting with 0) and hexadecimal (starting with 0x) are supported. Examples:

- "42" -> 42
- "042" -> 34 (octal -> decimal)
- "0x42" -> 66 (hexa -> decimal)

Warning: the conversion may overflow in various ways. Conversion is totally outsourced to strtol(), see the associated man page for overflow handling.

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

long integer

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

Supported values for integers include the usual C notation so decimal, octal (starting with 0) and hexadecimal (starting with 0x) are supported. Examples:

"42" -> 42 "042" -> 34 (octal -> decimal) "0x42" -> 66 (hexa -> decimal)

Warning: the conversion may overflow in various ways. Conversion is totally outsourced to strtol(), see the associated man page for overflow handling.

Credits: Thanks to A. Becker for suggesting strtol()

7.8.2.11 iniparser_getnsec()

```
int iniparser_getnsec (
            const dictionary * d )
```

Get number of sections in a dictionary.

Parameters

d	Dictionary to examine

Returns

int Number of sections found in dictionary

This function returns the number of sections found in a dictionary. The test to recognize sections is done on the string stored in the dictionary: a section name is given as "section" whereas a key is stored as "section:key", thus the test looks for entries that do not contain a colon.

This clearly fails in the case a section name contains a colon, but this should simply be avoided.

This function returns -1 in case of error.

7.8.2.12 iniparser_getseckeys()

```
const char** iniparser_getseckeys (
            const dictionary * d,
            const char * s,
            const char ** keys )
```

Get the number of keys in a section of a dictionary.

Parameters

d	Dictionary to examine
s	Section name of dictionary to examine
keys	Already allocated array to store the keys in

Returns

The pointer passed as `keys` argument or NULL in case of error

This function queries a dictionary and finds all keys in a given section. The keys argument should be an array of pointers which size has been determined by calling `iniparser_getsecnkeys` function prior to this one.

Each pointer in the returned char pointer-to-pointer is pointing to a string allocated in the dictionary; do not free or modify them.

7.8.2.13 iniparser_getsecname()

```
const char* iniparser_getsecname (
             const dictionary * d,
             int n )
```

Get name for section n in a dictionary.

Parameters

d	Dictionary to examine
n	Section number (from 0 to nsec-1).

Returns

Pointer to char string

This function locates the n-th section in a dictionary and returns its name as a pointer to a string statically allocated inside the dictionary. Do not free or modify the returned string!

This function returns NULL in case of error.

7.8.2.14 iniparser_getsecnkeys()

```
int iniparser_getsecnkeys (
             const dictionary * d,
             const char * s )
```

Get the number of keys in a section of a dictionary.

Parameters

d	Dictionary to examine
s	Section name of dictionary to examine

Returns

Number of keys in section

7.8.2.15 iniparser_getstring()

```
const char* iniparser_getstring (
            const dictionary * d,
            const char * key,
            const char * def )
```

Get the string associated to a key.

Parameters

d	Dictionary to search
key	Key string to look for
def	Default value to return if key not found.

Returns

pointer to statically allocated character string

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the pointer passed as 'def' is returned. The returned char pointer is pointing to a string allocated in the dictionary, do not free or modify it.

7.8.2.16 iniparser_load()

```
dictionary* iniparser_load (
            const char * ininame )
```

Parse an ini file and return an allocated dictionary object.

Parameters

ininame	Name of the ini file to read.

Returns

Pointer to newly allocated dictionary

This is the parser for ini files. This function is called, providing the name of the file to be read. It returns a dictionary object that should not be accessed directly, but through accessor functions instead.

The returned dictionary must be freed using iniparser_freedict().

7.8.2.17 iniparser_set()

```
int iniparser_set (
            dictionary * ini,
            const char * entry,
            const char * val )
```

Set an entry in a dictionary.

Parameters

ini	Dictionary to modify.
entry	Entry to modify (entry name)
val	New value to associate to the entry.

Returns

int 0 if Ok, -1 otherwise.

If the given entry can be found in the dictionary, it is modified to contain the provided value. If it cannot be found, the entry is created. It is Ok to set val to NULL.

7.8.2.18 iniparser_set_error_callback()

```
void iniparser_set_error_callback (
             int(*)(const char *,...)  errback )
```

Configure a function to receive the error messages.

Parameters

errback	Function to call.

By default, the error will be printed on stderr. If a null pointer is passed as errback the error callback will be switched back to default.

7.8.2.19 iniparser_unset()

```
void iniparser_unset (
             dictionary * ini,
             const char * entry )
```

Delete an entry in a dictionary.

Parameters

ini	Dictionary to modify
entry	Entry to delete (entry name)

Returns

void

If the given entry can be found, it is deleted from the dictionary.

7.9 palette.c File Reference

A collection of functions in aid of the care and feeding of palettes.

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include "palette.h"
#include "colors.h"
#include "cspace.h"
```

Macros

- #define **__USE_MINGW_ANSI_STDIO** 1

Functions

- void initArray (Array *a, size_t initialSize)

  Initialize Dynamic Array.
- void insertArray (Array *a, Rgb element)

  Add item to dynamic array.
- void freeArray (Array *a)

  Deallocate array.
- Array getColors (char *palname)

  Initialize palette from a palette file.
- Array setColors (Parameters p)

  Initialize palette from command line information.
- Array fromStart2Finish (Rgb start, Rgb finish, int N)

  Given a start Rgb value and a finish Rgb value, create a palette of N steps.
- void writeColors (Array colors, char *filename)

  Write out an image of the palette.
- Array readColors (char *filename)

  Read a palette file.
- Array buildColors (Parameters p)

  Build a palette from list of Rgb information.
- Rgb parseRGB (char *s, int *n)

  Mini parser for -k option format.
- void appendColors (Rgb start, Rgb finish, int N, Array *colors)

  Append a range of colors to a palette.

7.9.1 Detailed Description

A collection of functions in aid of the care and feeding of palettes.

Author

Hugh S. Myers

Date

Fri Aug 11 07:38:08 2017

7.9.2 DESCRIPTION

mp uses two methods to color a given fractal image. The first is a simple color table look-up, from a an array of Rgb values. This structure is refereed to as a palette, as in palette of colors like that used by artists. The functions here allow the creation, destruction, and manipulation of palettes.

Bug No known bugs.

Todo Refactor code to eliminate duplication and near-duplication.

7.9.3 Function Documentation

7.9.3.1 appendColors()

```
void appendColors (
            Rgb start,
            Rgb finish,
            int N,
            Array * colors )
```

Append a range of colors to a palette.

This function creates a range of colors between a pair of values, by stepping through HSV space and then converting each result to Rgb. The colors are created with a simple linear interpolation for each value.

Note

The apparent off-by-one error where the index of values starts at 1 rather than at zero is intentional. It prevents doubling the end/start value that would otherwise occur.

Parameters

start	First color in the sequence of Rgb values.
finish	Last color in the sequence of Rgb values.
N	number of steps in the interval between start and finish.
colors	existing dynamic array of colors to extend.

7.9.3.2 buildColors()

```
Array buildColors (
            Parameters p )
```

Build a palette from list of Rgb information.

Handle multiple -k entries and construct a palette by parsing the entries. The palette consists of colors laid end to end with each color specified acting as the head and tail of a color segment. Start with first, interpolate for listed (or default) number of values. Next segment starts with previous closing color and proceeds in the same fashion to the next color.

Parameters

p	Parameters file which contains stack of entries from the -k options

Returns

Returns a dynamic palette array.

7.9.3.3 freeArray()

```
void freeArray (
            Array * a )
```

Deallocate array.

Free memory allocated for dynamic array.

Parameters

a	A pointer to an array.

7.9.3.4 fromStart2Finish()

```
Array fromStart2Finish (
            Rgb start,
            Rgb finish,
            int N )
```

Given a start Rgb value and a finish Rgb value, create a palette of N steps.

Convert the given bracketing colors to HSV space and linearly interpolate between them.

Parameters

start	First or starting Rgb color.
finish	Second or finishing Rgb color.
N	Number of color values between start and finish (inclusive).

Returns

Returns an array of Rgb values, i.e. a palette.

7.9.3.5 getColors()

```
Array getColors (
            char * palname )
```

Initialize palette from a palette file.

Given the name of a palette file of Rgb triples read, create, and load a palette.

Parameters

palname	Name of palette file.

Returns

Returns an instance of dynamic array. Palette in this case.

7.9.3.6 initArray()

```
void initArray (
            Array * a,
            size_t initialSize )
```

Initialize Dynamic Array.

This begins a chain of memory allocations for a given typedef/struct. In this instance, the type is Rgb.

Parameters

A	pointer to an array.
initialSize	How many instances of item for inital allocation

7.9.3.7 insertArray()

```
void insertArray (
            Array * a,
            Rgb element )
```

Add item to dynamic array.

Either insert item into already allocated space or fetch more space and then insert.

Parameters

A	pointer to an array.
element	Item to be inserted.

7.9.3.8 parseRGB()

```
Rgb parseRGB (
            char * s,
            int * n )
```

Mini parser for -k option format.

-k or –kolor allows Rgb references in tow forms with an optional step indicator allowed in either case. If the reference begins with a '{' it is assumed to be an actual Rgb triplet and the the 3 values are scanned and parsed accordingly. If not a curly brace, then the assumption becomes that the entry is a named reference and a color name lookup is performed. Then the parser determines if a colon is present and if so, the numeric value is parsed and returned by setting the int pointer 'n' to the retrieved value. It defaults to 128. Finally the routine returns a palette.

Parameters

s	-k value.
n	pointer to an integer which receives the interval or step value.

Returns

Returns an Rgb triplet and sets the pointer to the interval.

7.9.3.9 readColors()

```
Array readColors (
            char * filename )
```

Read a palette file.

Read a palette file and create a palette, using the first line.

PPM File Format

1. A "magic number" for identifying the file type. A ppm image's magic number is the two characters "P6".
2. Whitespace (blanks, TABs, CRs, LFs).
3. A width, formatted as ASCII characters in decimal.
4. Whitespace.
5. A height, again in ASCII decimal.

6. Whitespace.
7. The maximum color value (Maxval), again in ASCII decimal. Must be less than 65536 and more than zero.
8. A single whitespace character (usually a newline).
9. A raster of Height rows, in order from top to bottom. Each row consists of Width pixels, in order from left to right. Each pixel is a triplet of red, green, and blue samples, in that order. Each sample is represented in pure binary by either 1 or 2 bytes. If the Maxval is less than 256, it is 1 byte. Otherwise, it is 2 bytes. The most significant byte is first.

Parameters

filename	Name of file to read.

Returns

Returns a dynamic palette array.

7.9.3.10 setColors()

```
Array setColors (
            Parameters p )
```

Initialize palette from command line information.

Given start and finish colors, construct a palette accordingly.

Parameters

p	Pointer to command line information typedef/struct

Returns

Returns an instance of dynamic array. Palette in this case.

7.9.3.11 writeColors()

```
void writeColors (
            Array colors,
            char * filename )
```

Write out an image of the palette.

Given a filename create a .ppm image file, displaying the chosen palette.

Parameters

colors	Palate to use.
filename	Name of file to create and write to,

7.10 palette.h File Reference

A collection of functions in aid of palette instantiation, manipulation, and deletion.

```
#include <complex.h>
#include "getopt.h"
```

Data Structures

- struct Rgb
- struct Array

Macros

- #define **PPMREADBUFLEN** 256

Functions

- Array buildColors (Parameters p)

  Build a palette from list of Rgb information.
- Array fromStart2Finish (Rgb start, Rgb finish, int N)

  Given a start Rgb value and a finish Rgb value, create a palette of N steps.
- Array getColors (char ∗palname)

  Initialize palette from a palette file.
- Array readColors (char ∗filename)

  Read a palette file.
- Array setColors (Parameters p)

  Initialize palette from command line information.
- Rgb parseRGB (char ∗s, int ∗n)

  Mini parser for -k option format.
- void appendColors (Rgb a, Rgb b, int interval, Array ∗color)

  Append a range of colors to a palette.
- void freeArray (Array ∗a)

  Deallocate array.
- void initArray (Array ∗a, size_t initialSize)

  Initialize Dynamic Array.
- void insertArray (Array ∗a, Rgb element)

  Add item to dynamic array.
- void writeColors (Array colors, char ∗filename)

  Write out an image of the palette.

7.10.1 Detailed Description

A collection of functions in aid of palette instantiation, manipulation, and deletion.

Author

Hugh S. Myers

Date

Sat Aug 12 07:32:49 2017

7.10.2 DESCRIPTION

palettes (color arrays) are collections of Rgb values and the associated functions here are the ways the palettes are used.

Bug No known bugs.

7.10.3 Data Structure Documentation

7.10.3.1 struct Rgb

Data Fields

unsigned char	b	
unsigned char	g	
unsigned char	r	

7.10.3.2 struct Array

Data Fields

Rgb *	array	
size_t	size	
size_t	used	

7.10.4 Function Documentation

7.10.4.1 appendColors()

```
void appendColors (
            Rgb start,
```

```
            Rgb finish,
            int N,
            Array * colors )
```

Append a range of colors to a palette.

This function creates a range of colors between a pair of values, by stepping through HSV space and then converting each result to Rgb. The colors are created with a simple linear interpolation for each value.

Note

The apparent off-by-one error where the index of values starts at 1 rather than at zero is intentional. It prevents doubling the end/start value that would otherwise occur.

Parameters

start	First color in the sequence of Rgb values.
finish	Last color in the sequence of Rgb values.
N	number of steps in the interval between start and finish.
colors	existing dynamic array of colors to extend.

7.10.4.2 buildColors()

```
Array buildColors (
            Parameters p )
```

Build a palette from list of Rgb information.

Handle multiple -k entries and construct a palette by parsing the entries. The palette consists of colors laid end to end with each color specified acting as the head and tail of a color segment. Start with first, interpolate for listed (or default) number of values. Next segment starts with previous closing color and proceeds in the same fashion to the next color.

Parameters

p	Parameters file which contains stack of entries from the -k options

Returns

Returns a dynamic palette array.

7.10.4.3 freeArray()

```
void freeArray (
            Array * a )
```

Deallocate array.

Free memory allocated for dynamic array.

Parameters

a	A pointer to an array.

7.10.4.4 fromStart2Finish()

```
Array fromStart2Finish (
            Rgb start,
            Rgb finish,
            int N )
```

Given a start Rgb value and a finish Rgb value, create a palette of N steps.

Convert the given bracketing colors to HSV space and linearly interpolate between them.

Parameters

start	First or starting Rgb color.
finish	Second or finishing Rgb color.
N	Number of color values between start and finish (inclusive).

Returns

Returns an array of Rgb values, i.e. a palette.

7.10.4.5 getColors()

```
Array getColors (
            char * palname )
```

Initialize palette from a palette file.

Given the name of a palette file of Rgb triples read, create, and load a palette.

Parameters

palname	Name of palette file.

Returns

Returns an instance of dynamic array. Palette in this case.

7.10.4.6 initArray()

```
void initArray (
            Array * a,
            size_t initialSize )
```

Initialize Dynamic Array.

This begins a chain of memory allocations for a given typedef/struct. In this instance, the type is Rgb.

Parameters

A	pointer to an array.
initialSize	How many instances of item for inital allocation

7.10.4.7 insertArray()

```
void insertArray (
            Array * a,
            Rgb element )
```

Add item to dynamic array.

Either insert item into already allocated space or fetch more space and then insert.

Parameters

A	pointer to an array.
element	Item to be inserted.

7.10.4.8 parseRGB()

```
Rgb parseRGB (
            char * s,
            int * n )
```

Mini parser for -k option format.

-k or –kolor allows Rgb references in tow forms with an optional step indicator allowed in either case. If the reference begins with a '{' it is assumed to be an actual Rgb triplet and the the 3 values are scanned and parsed accordingly. If not a curly brace, then the assumption becomes that the entry is a named reference and a color name lookup is performed. Then the parser determines if a colon is present and if so, the numeric value is parsed and returned by setting the int pointer 'n' to the retrieved value. It defaults to 128. Finally the routine returns a palette.

Parameters

s	-k value.
n	pointer to an integer which receives the interval or step value.

Returns

Returns an Rgb triplet and sets the pointer to the interval.

7.10.4.9 readColors()

```
Array readColors (
            char * filename )
```

Read a palette file.

Read a palette file and create a palette, using the first line.

PPM File Format

1. A "magic number" for identifying the file type. A ppm image's magic number is the two characters "P6".
2. Whitespace (blanks, TABs, CRs, LFs).
3. A width, formatted as ASCII characters in decimal.
4. Whitespace.
5. A height, again in ASCII decimal.
6. Whitespace.
7. The maximum color value (Maxval), again in ASCII decimal. Must be less than 65536 and more than zero.
8. A single whitespace character (usually a newline).
9. A raster of Height rows, in order from top to bottom. Each row consists of Width pixels, in order from left to right. Each pixel is a triplet of red, green, and blue samples, in that order. Each sample is represented in pure binary by either 1 or 2 bytes. If the Maxval is less than 256, it is 1 byte. Otherwise, it is 2 bytes. The most significant byte is first.

Parameters

filename	Name of file to read.

Returns

Returns a dynamic palette array.

7.10.4.10 setColors()

```
Array setColors (
            Parameters p )
```

Initialize palette from command line information.

Given start and finish colors, construct a palette accordingly.

Parameters

p	Pointer to command line information typedef/struct

Returns

Returns an instance of dynamic array. Palette in this case.

7.10.4.11 writeColors()

```
void writeColors (
            Array colors,
            char * filename )
```

Write out an image of the palette.

Given a filename create a .ppm image file, displaying the chosen palette.

Parameters

colors	Palate to use.
filename	Name of file to create and write to,

7.11 util.c File Reference

A collection of functions in aid of support for mp.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include <quadmath.h>
#include <ctype.h>
#include "util.h"
```

Functions

- char * getCl (int argc, char *argv[])

  Assemble the command-line as a string.
- char * **ReadFile** (char *filename)
- double **scaleWidth** (double d, int N, double E)
- int **bestGuess** (double diameter, int W)
- char * **guessStr** (int g)
- void **str2abbr** (char *abbr, size_t size, const char *s)
- __float128 **fabsq** (__float128 x)
- int **countChar** (const char *s, const char c)
- void **signon** (const char *prog, const char *Version)

7.11.1 Detailed Description

A collection of functions in aid of support for mp.

Author

Hugh S. Myers

Date

Mon Jul 24 17:01:02 2017

General catch-all of routines needed but not complicated enough for their own file.

Bug No known bugs.

7.11.2 Function Documentation

7.11.2.1 getCl()

```
char* getCl (
              int argc,
              char * argv[] )
```

Assemble the command-line as a string.

Reconstruct the command-line from argc and argv.

Parameters

argc	Argument count.
argv	Command-line substrings array.

Returns

An image of the original command-line as a pointer to a static buffer.

7.12 util.h File Reference

A collection of functions in aid of this and that.

```
#include <math.h>
#include <complex.h>
```

Macros

- #define **max**(x, y)

Enumerations

- enum {
  FLT, **DBL**, **LDBL**, **FLT128**,
  FLTMPC, **NOTYET** }

Functions

- __float128 **fabsq** (__float128 x)
- char * getCl (int argc, char *argv[])

  Assemble the command-line as a string.
- char * **guessStr** (int g)
- char * **ReadFile** (char *filename)
- double **scaleWidth** (double d, int N, double E)
- int **bestGuess** (double diameter, int W)
- int **countChar** (const char *s, const char c)
- void **signon** (const char *prog, const char *Version)
- void **str2abbr** (char *abbr, size_t size, const char *s)

7.12.1 Detailed Description

A collection of functions in aid of this and that.

Author

Hugh S. Myers

Date

Mon Jul 24 19:30:12 2017

7.12.2 DESCRIPTION

Convenient parking lot for utility functions used by mp.

Bug No known bugs.

7.12.3 Macro Definition Documentation

7.12.3.1 max

```
#define max(
               x,
               y )
```

Value:

```
( \
    { __auto_type __x = (x); __auto_type __y = (y); \
      __x > __y ? __x : __y; })
```

7.12.4 Function Documentation

7.12.4.1 getCl()

```
char* getCl (
             int argc,
             char * argv[] )
```

Assemble the command-line as a string.

Reconstruct the command-line from argc and argv.

Parameters

argc	Argument count.
argv	Command-line substrings array.

Returns

An image of the original command-line as a pointer to a static buffer.

Index