rsstats

Generated by Doxygen 1.9.4

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

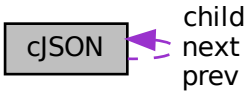
Chapter 3

Class Documentation

3.1 cJSON Struct Reference

`#include <cJSON.h>`

Collaboration diagram for cJSON:



Public Attributes

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

3.1.1 Detailed Description

Definition at line 103 of file cJSON.h.

3.1.2 Member Data Documentation

3.1.2.1 child

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

Definition at line 108 of file cJSON.h.

3.1.2.2 next

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

Definition at line 105 of file cJSON.h.

3.1.2.3 prev

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

Definition at line 106 of file cJSON.h.

3.1.2.4 string

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

Definition at line 121 of file cJSON.h.

3.1.2.5 type

```
int cJSON::type
```

Definition at line 111 of file cJSON.h.

3.1.2.6 valuedouble

```
double cJSON::valuedouble
```

Definition at line 118 of file cJSON.h.

3.1.2.7 valueint

```
int cJSON::valueint
```

Definition at line 116 of file cJSON.h.

3.1.2.8 valuestring

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

Definition at line 114 of file cJSON.h.

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

- cJSON.h

3.2 cJSON_Hooks Struct Reference

```
#include <cJSON.h>
```

Public Member Functions

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

3.2.1 Detailed Description

Definition at line 124 of file cJSON.h.

3.2.2 Member Function Documentation

3.2.2.1 malloc_fn()

```
void *CJSON_CDECL * cJSON_Hooks::malloc_fn (
            size_t sz )
```

3.2.2.2 void()

```
cJSON_Hooks::void (
            CJSON_CDECL * free_fn )
```

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

- cJSON.h

3.3 cluster_s Struct Reference

```
#include <cluster.h>
```

Public Attributes

- unsigned short int enabled
- char * host
- char * user
- char * pass
- char * insecure
- char * cacert

3.3.1 Detailed Description

Definition at line 24 of file cluster.h.

3.3.2 Member Data Documentation

3.3.2.1 cacert

```
char* cluster_s::cacert
```

Definition at line 30 of file cluster.h.

3.3.2.2 enabled

```
unsigned short int cluster_s::enabled
```

Definition at line 25 of file cluster.h.

3.3.2.3 host

```
char* cluster_s::host
```

Definition at line 26 of file cluster.h.

3.3.2.4 insecure

```
char* cluster_s::insecure
```

Definition at line 29 of file cluster.h.

3.3.2.5 pass

```
char* cluster_s::pass
```

Definition at line 28 of file cluster.h.

3.3.2.6 user

```
char* cluster_s::user
```

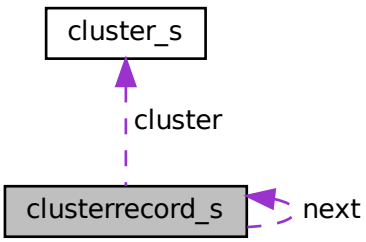
Definition at line 27 of file cluster.h.

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

- cluster.h

3.4 clusterrecord_s Struct Reference

Collaboration diagram for clusterrecord_s:



Public Attributes

- struct clusterrecord_s * next
- cluster_t * cluster

3.4.1 Detailed Description

Definition at line 31 of file clusterlst.c.

3.4.2 Member Data Documentation

3.4.2.1 cluster

```
cluster_t* clusterrecord_s::cluster
```

Definition at line 33 of file clusterlst.c.

3.4.2.2 next

```
struct clusterrecord_s* clusterrecord_s::next
```

Definition at line 32 of file clusterlst.c.

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

- clusterlst.c

3.5 error Struct Reference

Public Attributes

- const unsigned char * json
- size_t position

3.5.1 Detailed Description

Definition at line 88 of file cJSON.c.

3.5.2 Member Data Documentation

3.5.2.1 json

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

Definition at line 89 of file cJSON.c.

3.5.2.2 position

```
size_t error::position
```

Definition at line 90 of file cJSON.c.

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

- cJSON.c

3.6 internal_hooks Struct Reference

Public Member Functions

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

3.6.1 Detailed Description

Definition at line 145 of file cJSON.c.

3.6.2 Member Function Documentation

3.6.2.1 allocate()

```
void *CJSON_CDECL * internal_hooks::allocate (
            size_t size )
```

3.6.2.2 reallocate()

```
void *CJSON_CDECL * internal_hooks::reallocate (
            void * pointer,
            size_t size )
```

3.6.2.3 void()

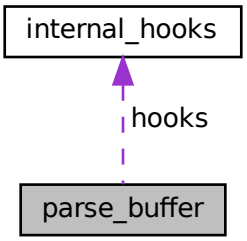
```
internal_hooks::void (
            CJSON_CDECL * deallocate )
```

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

- cJSON.c

3.7 parse_buffer Struct Reference

Collaboration diagram for parse_buffer:



Public Attributes

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

3.7.1 Detailed Description

Definition at line 258 of file cJSON.c.

3.7.2 Member Data Documentation

3.7.2.1 content

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

Definition at line 259 of file cJSON.c.

3.7.2.2 depth

```
size_t parse_buffer::depth
```

Definition at line 262 of file cJSON.c.

3.7.2.3 hooks

```
internal_hooks parse_buffer::hooks
```

Definition at line 263 of file cJSON.c.

3.7.2.4 length

```
size_t parse_buffer::length
```

Definition at line 260 of file cJSON.c.

3.7.2.5 offset

```
size_t parse_buffer::offset
```

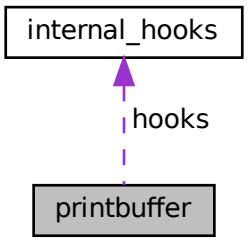
Definition at line 261 of file cJSON.c.

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

- cJSON.c

3.8 printbuffer Struct Reference

Collaboration diagram for printbuffer:



Public Attributes

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

3.8.1 Detailed Description

Definition at line 391 of file cJSON.c.

3.8.2 Member Data Documentation

3.8.2.1 buffer

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

Definition at line 392 of file cJSON.c.

3.8.2.2 depth

```
size_t printbuffer::depth
```

Definition at line 395 of file cJSON.c.

3.8.2.3 format

```
cJSON_bool printbuffer::format
```

Definition at line 397 of file cJSON.c.

3.8.2.4 hooks

```
internal_hooks printbuffer::hooks
```

Definition at line 398 of file cJSON.c.

3.8.2.5 length

```
size_t printbuffer::length
```

Definition at line 393 of file cJSON.c.

3.8.2.6 noalloc

```
cJSON_bool printbuffer::noalloc
```

Definition at line 396 of file cJSON.c.

3.8.2.7 offset

```
size_t printbuffer::offset
```

Definition at line 394 of file cJSON.c.

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

- cJSON.c

3.9 rsclustercon_s Struct Reference

`#include <clustercon.h>`

Public Attributes

- char * host
- char * user
- char * pass
- unsigned short int insecure
- char * cacert
- int sock
- SSL_CTX * ctx
- SSL * ssl

3.9.1 Detailed Description

Definition at line 29 of file clustercon.h.

3.9.2 Member Data Documentation

3.9.2.1 cacert

`char* rsclustercon_s::cacert`

Definition at line 34 of file clustercon.h.

3.9.2.2 ctx

`SSL_CTX* rsclustercon_s::ctx`

Definition at line 36 of file clustercon.h.

3.9.2.3 host

`char* rsclustercon_s::host`

Definition at line 30 of file clustercon.h.

3.9.2.4 insecure

```
unsigned short int rsclustercon_s::insecure
```

Definition at line 33 of file clustercon.h.

3.9.2.5 pass

```
char* rsclustercon_s::pass
```

Definition at line 32 of file clustercon.h.

3.9.2.6 sock

```
int rsclustercon_s::sock
```

Definition at line 35 of file clustercon.h.

3.9.2.7 ssl

```
SSL* rsclustercon_s::ssl
```

Definition at line 37 of file clustercon.h.

3.9.2.8 user

```
char* rsclustercon_s::user
```

Definition at line 31 of file clustercon.h.

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

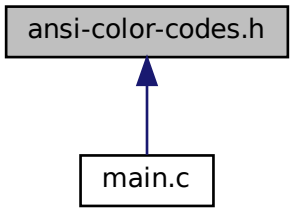
- clustercon.h

Chapter 4

File Documentation

4.1 ansi-color-codes.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define BLK "\33[0;30m"
- #define RED "\33[0;31m"
- #define GRN "\33[0;32m"
- #define YEL "\33[0;33m"
- #define BLU "\33[0;34m"
- #define MAG "\33[0;35m"
- #define CYN "\33[0;36m"
- #define WHT "\33[0;37m"
- #define BBLK "\33[1;30m"
- #define BRED "\33[1;31m"
- #define BGRN "\33[1;32m"
- #define BYEL "\33[1;33m"
- #define BBLU "\33[1;34m"
- #define BMAG "\33[1;35m"
- #define BCYN "\33[1;36m"
- #define BWHT "\33[1;37m"

- #define UBLK "\33[4;30m"
- #define URED "\33[4;31m"
- #define UGRN "\33[4;32m"
- #define UYEL "\33[4;33m"
- #define UBLU "\33[4;34m"
- #define UMAG "\33[4;35m"
- #define UCYN "\33[4;36m"
- #define UWHT "\33[4;37m"
- #define BLKB "\33[40m"
- #define REDB "\33[41m"
- #define GRNB "\33[42m"
- #define YELB "\33[43m"
- #define BLUB "\33[44m"
- #define MAGB "\33[45m"
- #define CYNB "\33[46m"
- #define WHTB "\33[47m"
- #define BLKHB "\33[0;100m"
- #define REDHB "\33[0;101m"
- #define GRNHB "\33[0;102m"
- #define YELHB "\33[0;103m"
- #define BLUHB "\33[0;104m"
- #define MAGHB "\33[0;105m"
- #define CYNHB "\33[0;106m"
- #define WHTHB "\33[0;107m"
- #define HBLK "\33[0;90m"
- #define HRED "\33[0;91m"
- #define HGRN "\33[0;92m"
- #define HYEL "\33[0;93m"
- #define HBLU "\33[0;94m"
- #define HMAG "\33[0;95m"
- #define HCYN "\33[0;96m"
- #define HWHT "\33[0;97m"
- #define BHBLK "\33[1;90m"
- #define BHRED "\33[1;91m"
- #define BHGRN "\33[1;92m"
- #define BHYEL "\33[1;93m"
- #define BHBLU "\33[1;94m"
- #define BHMAG "\33[1;95m"
- #define BHCYN "\33[1;96m"
- #define BHWHT "\33[1;97m"
- #define RESET "\33[0m"
- #define DIM "\33[22m"
- #define BLINK "\33[5m"
- #define HIDDEN "\33[8m"
- #define REVERSE "\33[7m"
- #define BOLD "\33[1m"
- #define UNDERLINE "\33[4m"
- #define STRIKE "\33[9m"

4.1.1 Macro Definition Documentation

4.1.1.1 BBLK

```
#define BBLK "\33[1;30m"
```

Definition at line 20 of file ansi-color-codes.h.

4.1.1.2 BBLU

```
#define BBLU "\33[1;34m"
```

Definition at line 24 of file ansi-color-codes.h.

4.1.1.3 BCYN

```
#define BCYN "\33[1;36m"
```

Definition at line 26 of file ansi-color-codes.h.

4.1.1.4 BGRN

```
#define BGRN "\33[1;32m"
```

Definition at line 22 of file ansi-color-codes.h.

4.1.1.5 BHBLK

```
#define BHBLK "\33[1;90m"
```

Definition at line 70 of file ansi-color-codes.h.

4.1.1.6 BHBLU

```
#define BHBLU "\33[1;94m"
```

Definition at line 74 of file ansi-color-codes.h.

4.1.1.7 BHCYN

```
#define BHCYN "\33[1;96m"
```

Definition at line 76 of file ansi-color-codes.h.

4.1.1.8 BHGRN

```
#define BHGRN "\33[1;92m"
```

Definition at line 72 of file ansi-color-codes.h.

4.1.1.9 BHMAG

```
#define BHMAG "\33[1;95m"
```

Definition at line 75 of file ansi-color-codes.h.

4.1.1.10 BHRED

```
#define BHRED "\33[1;91m"
```

Definition at line 71 of file ansi-color-codes.h.

4.1.1.11 BHWHT

```
#define BHWHT "\33[1;97m"
```

Definition at line 77 of file ansi-color-codes.h.

4.1.1.12 BHYEL

```
#define BHYEL "\33[1;93m"
```

Definition at line 73 of file ansi-color-codes.h.

4.1.1.13 BLINK

```
#define BLINK "\33[5m"
```

Definition at line 82 of file ansi-color-codes.h.

4.1.1.14 BLK

```
#define BLK "\33[0;30m"
```

Definition at line 10 of file ansi-color-codes.h.

4.1.1.15 BLKB

```
#define BLKB "\33[40m"
```

Definition at line 40 of file ansi-color-codes.h.

4.1.1.16 BLKHB

```
#define BLKHB "\33[0;100m"
```

Definition at line 50 of file ansi-color-codes.h.

4.1.1.17 BLU

```
#define BLU "\33[0;34m"
```

Definition at line 14 of file ansi-color-codes.h.

4.1.1.18 BLUB

```
#define BLUB "\33[44m"
```

Definition at line 44 of file ansi-color-codes.h.

4.1.1.19 BLUHB

```
#define BLUHB "\33[0;104m"
```

Definition at line 54 of file ansi-color-codes.h.

4.1.1.20 BMAG

```
#define BMAG "\33[1;35m"
```

Definition at line 25 of file ansi-color-codes.h.

4.1.1.21 BOLD

```
#define BOLD "\33[1m"
```

Definition at line 85 of file ansi-color-codes.h.

4.1.1.22 BRED

```
#define BRED "\33[1;31m"
```

Definition at line 21 of file ansi-color-codes.h.

4.1.1.23 BWHT

```
#define BWHT "\33[1;37m"
```

Definition at line 27 of file ansi-color-codes.h.

4.1.1.24 BYEL

```
#define BYEL "\33[1;33m"
```

Definition at line 23 of file ansi-color-codes.h.

4.1.1.25 CYN

```
#define CYN "\33[0;36m"
```

Definition at line 16 of file ansi-color-codes.h.

4.1.1.26 CYNB

```
#define CYNB "\33[46m"
```

Definition at line 46 of file ansi-color-codes.h.

4.1.1.27 CYNHB

```
#define CYNHB "\33[0;106m"
```

Definition at line 56 of file ansi-color-codes.h.

4.1.1.28 DIM

```
#define DIM "\33[22m"
```

Definition at line 81 of file ansi-color-codes.h.

4.1.1.29 GRN

```
#define GRN "\33[0;32m"
```

Definition at line 12 of file ansi-color-codes.h.

4.1.1.30 GRNB

```
#define GRNB "\33[42m"
```

Definition at line 42 of file ansi-color-codes.h.

4.1.1.31 GRNHB

```
#define GRNHB "\33[0;102m"
```

Definition at line 52 of file ansi-color-codes.h.

4.1.1.32 HBLK

```
#define HBLK "\33[0;90m"
```

Definition at line 60 of file ansi-color-codes.h.

4.1.1.33 HBLU

```
#define HBLU "\33[0;94m"
```

Definition at line 64 of file ansi-color-codes.h.

4.1.1.34 HCYN

```
#define HCYN "\33[0;96m"
```

Definition at line 66 of file ansi-color-codes.h.

4.1.1.35 HGRN

```
#define HGRN "\33[0;92m"
```

Definition at line 62 of file ansi-color-codes.h.

4.1.1.36 HIDDEN

```
#define HIDDEN "\33[8m"
```

Definition at line 83 of file ansi-color-codes.h.

4.1.1.37 HMAG

```
#define HMAG "\33[0;95m"
```

Definition at line 65 of file ansi-color-codes.h.

4.1.1.38 HRED

```
#define HRED "\33[0;91m"
```

Definition at line 61 of file ansi-color-codes.h.

4.1.1.39 HWHT

```
#define HWHT "\33[0;97m"
```

Definition at line 67 of file ansi-color-codes.h.

4.1.1.40 HYEL

```
#define HYEL "\33[0;93m"
```

Definition at line 63 of file ansi-color-codes.h.

4.1.1.41 MAG

```
#define MAG "\33[0;35m"
```

Definition at line 15 of file ansi-color-codes.h.

4.1.1.42 MAGB

```
#define MAGB "\33[45m"
```

Definition at line 45 of file ansi-color-codes.h.

4.1.1.43 MAGHB

```
#define MAGHB "\33[0;105m"
```

Definition at line 55 of file ansi-color-codes.h.

4.1.1.44 RED

```
#define RED "\33[0;31m"
```

Definition at line 11 of file ansi-color-codes.h.

4.1.1.45 REDB

```
#define REDB "\33[41m"
```

Definition at line 41 of file ansi-color-codes.h.

4.1.1.46 REDHB

```
#define REDHB "\33[0;101m"
```

Definition at line 51 of file ansi-color-codes.h.

4.1.1.47 RESET

```
#define RESET "\33[0m"
```

Definition at line 80 of file ansi-color-codes.h.

4.1.1.48 REVERSE

```
#define REVERSE "\33[7m"
```

Definition at line 84 of file ansi-color-codes.h.

4.1.1.49 STRIKE

```
#define STRIKE "\33[9m"
```

Definition at line 87 of file ansi-color-codes.h.

4.1.1.50 UBLK

```
#define UBLK "\33[4;30m"
```

Definition at line 30 of file ansi-color-codes.h.

4.1.1.51 UBLU

```
#define UBLU "\33[4;34m"
```

Definition at line 34 of file ansi-color-codes.h.

4.1.1.52 UCYN

```
#define UCYN "\33[4;36m"
```

Definition at line 36 of file ansi-color-codes.h.

4.1.1.53 UGRN

```
#define UGRN "\33[4;32m"
```

Definition at line 32 of file ansi-color-codes.h.

4.1.1.54 UMAG

```
#define UMAG "\33[4;35m"
```

Definition at line 35 of file ansi-color-codes.h.

4.1.1.55 UNDERLINE

```
#define UNDERLINE "\33[4m"
```

Definition at line 86 of file ansi-color-codes.h.

4.1.1.56 URED

```
#define URED "\33[4;31m"
```

Definition at line 31 of file ansi-color-codes.h.

4.1.1.57 UWHT

```
#define UWHT "\33[4;37m"
```

Definition at line 37 of file ansi-color-codes.h.

4.1.1.58 UYEL

```
#define UYEL "\33[4;33m"
```

Definition at line 33 of file ansi-color-codes.h.

4.1.1.59 WHT

```
#define WHT "\33[0;37m"
```

Definition at line 17 of file ansi-color-codes.h.

4.1.1.60 WHTB

```
#define WHTB "\33[47m"
```

Definition at line 47 of file ansi-color-codes.h.

4.1.1.61 WHTHB

```
#define WHTHB "\33[0;107m"
```

Definition at line 57 of file ansi-color-codes.h.

4.1.1.62 YEL

```
#define YEL "\33[0;33m"
```

Definition at line 13 of file ansi-color-codes.h.

4.1.1.63 YELB

```
#define YELB "\33[43m"
```

Definition at line 43 of file ansi-color-codes.h.

4.1.1.64 YELHB

```
#define YELHB "\33[0;103m"
```

Definition at line 53 of file ansi-color-codes.h.

4.2 ansi-color-codes.h

Go to the documentation of this file.

```
00001 /*
00002 * This is free and unencumbered software released into the public domain.
00003 *
00004 * For more information, please refer to <https://unlicense.org>
00005 *
00006 * Downloaded from https://gist.github.com/federicheddu/036ddc1624c12c073d1d481f3044628a
00007 */
00008
00009 /* Regular text */
00010 #define BLK "\33[0;30m"
00011 #define RED "\33[0;31m"
00012 #define GRN "\33[0;32m"
00013 #define YEL "\33[0;33m"
00014 #define BLU "\33[0;34m"
00015 #define MAG "\33[0;35m"
00016 #define CYN "\33[0;36m"
00017 #define WHT "\33[0;37m"
00018
00019 /* Regular bold text */
00020 #define BBLK "\33[1;30m"
00021 #define BRED "\33[1;31m"
00022 #define BGRN "\33[1;32m"
00023 #define BYEL "\33[1;33m"
00024 #define BBLU "\33[1;34m"
00025 #define BMAG "\33[1;35m"
00026 #define BCYN "\33[1;36m"
```

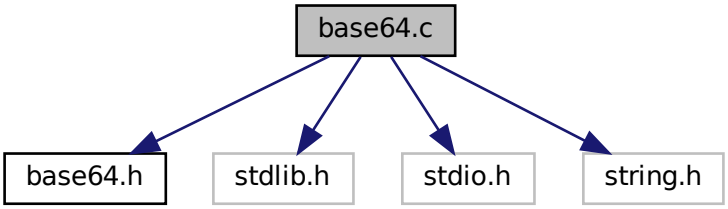
```
00027 #define BWHT "\33[1;37m"
00028
00029 /* Regular underline text */
00030 #define UBLK "\33[4;30m"
00031 #define URED "\33[4;31m"
00032 #define UGRN "\33[4;32m"
00033 #define UYEL "\33[4;33m"
00034 #define UBLU "\33[4;34m"
00035 #define UMAG "\33[4;35m"
00036 #define UCYN "\33[4;36m"
00037 #define UWHT "\33[4;37m"
00038
00039 /* Regular background */
00040 #define BLKB "\33[40m"
00041 #define REDB "\33[41m"
00042 #define GRNB "\33[42m"
00043 #define YELB "\33[43m"
00044 #define BLUB "\33[44m"
00045 #define MAGB "\33[45m"
00046 #define CYNB "\33[46m"
00047 #define WHTB "\33[47m"
00048
00049 /* High intensty background */
00050 #define BLKHB "\33[0;100m"
00051 #define REDHB "\33[0;101m"
00052 #define GRNHB "\33[0;102m"
00053 #define YELHB "\33[0;103m"
00054 #define BLUHB "\33[0;104m"
00055 #define MAGHB "\33[0;105m"
00056 #define CYNHB "\33[0;106m"
00057 #define WHTHB "\33[0;107m"
00058
00059 /* High intensty text */
00060 #define HBLK "\33[0;90m"
00061 #define HRED "\33[0;91m"
00062 #define HGRN "\33[0;92m"
00063 #define HYEL "\33[0;93m"
00064 #define HBLU "\33[0;94m"
00065 #define HMAG "\33[0;95m"
00066 #define HCYN "\33[0;96m"
00067 #define HWHT "\33[0;97m"
00068
00069 /* Bold high intensity text */
00070 #define BHBLK "\33[1;90m"
00071 #define BHRED "\33[1;91m"
00072 #define BHGRN "\33[1;92m"
00073 #define BHYEL "\33[1;93m"
00074 #define BHBLU "\33[1;94m"
00075 #define BHMAG "\33[1;95m"
00076 #define BHCYN "\33[1;96m"
00077 #define BHWHT "\33[1;97m"
00078
00079 /* Reset */
00080 #define RESET "\33[0m"
00081 #define DIM "\33[22m"
00082 #define BLINK "\33[5m"
00083 #define HIDDEN "\33[8m"
00084 #define REVERSE "\33[7m"
00085 #define BOLD "\33[1m"
00086 #define UNDERLINE "\33[4m"
00087 #define STRIKE "\33[9m"
00088
00089 /* vim:  set tw=80:  */
```

4.3 base64.c File Reference

Simple Base64 encoding and decoding functions.

```
#include "base64.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

Include dependency graph for base64.c:



Functions

- char * base64_encode (char *plain)

  Encode a zero terminated C string in Base64.
- char * base64_decode (char *cipher)

  Decode a zero terminated C Base64 encoded string.

4.3.1 Detailed Description

Simple Base64 encoding and decoding functions.

Copied and adapted from https://github.com/elzoughby/Base64

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file base64.c.

4.3.2 Function Documentation

4.3.2.1 base64_decode()

```
char * base64_decode (
            char * cipher )
```

Decode a zero terminated C Base64 encoded string.

Parameters

cipher	Zero Terminated C Base64 encoded string

Returns

Decoded zero terminated C string

Return values

Pointer	to a zero terminated C string
NULL	in case of out of memory

cipher can not be NULL. the returned value is mallocated and needs to be freed.

Definition at line 77 of file base64.c.

4.3.2.2 base64_encode()

```
char * base64_encode (
            char * plain )
```

Encode a zero terminated C string in Base64.

Parameters

plain	Zero Terminated C string

Returns

Encoded zero terminated C string

Return values

Pointer	to a zero terminated C string
NULL	in case of out of memory

plain can not be NULL. the returned value is mallocated and needs to be freed.

Definition at line 39 of file base64.c.

Here is the caller graph for this function:

4.4 base64.c

Go to the documentation of this file.

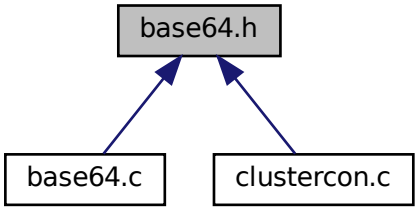
```
00001
00021 #ifdef HAVE_CONFIG_H
00022 #include "config.h"
00023 #endif
00024
00025 #include "base64.h"
00026
00027 #include <stdlib.h>
00028 #include <stdio.h>                              /* perror */
00029 #include <string.h>
00030
00031 static char base46_map[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
00032                             'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y',
00033                             'Z', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
00034                             'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '0', '1', '2',
00035                             '3', '4', '5', '6', '7', '8', '9', '+', '/'
00036                            };
00037
00038
00039 char* base64_encode(char* plain) {
00040     unsigned char counts = 0;
00041     char buffer[3];
00042     char* cipher = malloc(strlen(plain) * 4 / 3 + 4);
00043     int i = 0, c = 0;
00044
00045     if (NULL==cipher) {
00046         perror("base64_encode");
00047         return NULL;
00048     }
00049
00050     for(i = 0; plain[i] != '\0'; i++) {
00051         buffer[counts++] = plain[i];
00052         if(counts == 3) {
00053             cipher[c++] = base46_map[buffer[0] » 2];
00054             cipher[c++] = base46_map[((buffer[0] & 0x03) « 4) + (buffer[1] » 4)];
00055             cipher[c++] = base46_map[((buffer[1] & 0x0f) « 2) + (buffer[2] » 6)];
00056             cipher[c++] = base46_map[buffer[2] & 0x3f];
00057             counts = 0;
00058         }
00059     }
00060
00061     if(counts > 0) {
00062         cipher[c++] = base46_map[buffer[0] » 2];
00063         if(counts == 1) {
00064             cipher[c++] = base46_map[(buffer[0] & 0x03) « 4];
00065             cipher[c++] = '=';
00066         } else {                      // if counts == 2
00067             cipher[c++] = base46_map[((buffer[0] & 0x03) « 4) + (buffer[1] » 4)];
00068             cipher[c++] = base46_map[(buffer[1] & 0x0f) « 2];
00069         }
00070         cipher[c++] = '=';
00071     }
00072
00073     cipher[c] = '\0';   /* string padding character */
00074     return cipher;
00075 }
00076
00077 char* base64_decode(char* cipher) {
00078
00079     unsigned char counts = 0;
00080     char buffer[4];
```

```
00081     char* plain = malloc(strlen(cipher) * 3 / 4 + 1);
00082     int i = 0, p = 0;
00083
00084     if (NULL==plain) {
00085         perror("base64_decode");
00086         return NULL;
00087     }
00088
00089     for(i = 0; cipher[i] != '\0'; i++) {
00090         unsigned char k;
00091         for(k = 0 ; k < 64 && base46_map[k] != cipher[i]; k++);
00092         buffer[counts++] = k;
00093         if(counts == 4) {
00094             plain[p++] = (buffer[0] << 2) + (buffer[1] >> 4);
00095             if(buffer[2] != 64)
00096                 plain[p++] = (buffer[1] << 4) + (buffer[2] >> 2);
00097             if(buffer[3] != 64)
00098                 plain[p++] = (buffer[2] << 6) + buffer[3];
00099             counts = 0;
00100         }
00101     }
00102
00103     plain[p] = '\0';    /* string padding character */
00104     return plain;
00105 }
00106
00107 /* vim:  set tw=80:  */
```

4.5 base64.h File Reference

Simple Base64 encoding and decoding functions.

This graph shows which files directly or indirectly include this file:



Functions

- char * base64_encode (char *plain)

  Encode a zero terminated C string in Base64.
- char * base64_decode (char *cipher)

  Decode a zero terminated C Base64 encoded string.

4.5.1 Detailed Description

Simple Base64 encoding and decoding functions.

Copied and adapted from https://github.com/elzoughby/Base64

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file base64.h.

4.5.2 Function Documentation

4.5.2.1 base64_decode()

```
char * base64_decode (
            char * cipher )
```

Decode a zero terminated C Base64 encoded string.

Parameters

cipher	Zero Terminated C Base64 encoded string

Returns

Decoded zero terminated C string

Return values

Pointer	to a zero terminated C string
NULL	in case of out of memory

cipher can not be NULL. the returned value is mallocated and needs to be freed.

Definition at line 77 of file base64.c.

4.5.2.2 base64_encode()

```
char * base64_encode (
            char * plain )
```

Encode a zero terminated C string in Base64.

Parameters

plain	Zero Terminated C string

Returns

Encoded zero terminated C string

Return values

Pointer	to a zero terminated C string
NULL	in case of out of memory

plain can not be NULL. the returned value is mallocated and needs to be freed.

Definition at line 39 of file base64.c.

Here is the caller graph for this function:

4.6 base64.h

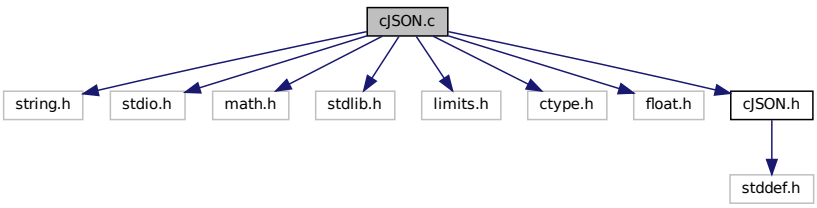
Go to the documentation of this file.

```
00001
00021 #ifndef __BASE64_H__
00022 #define __BASE64_H__
00023
00035 char* base64_encode(char* plain);
00036
00048 char* base64_decode(char* cipher);
00049
00050 #endif /* __BASE64_H__ */
00051
00052 /* vim:  set tw=80:  */
```

4.7 cJSON.c File Reference

```
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <limits.h>
#include <ctype.h>
#include <float.h>
#include "cJSON.h"
```

Include dependency graph for cJSON.c:



Classes

- struct error
- struct internal_hooks
- struct parse_buffer
- struct printbuffer

Macros

- #define true ((cJSON_bool)1)
- #define false ((cJSON_bool)0)
- #define isinf(d) (isnan((d - d)) && !isnan(d))
- #define isnan(d) (d != d)
- #define NAN 0.0/0.0
- #define internal_malloc malloc
- #define internal_free free
- #define internal_realloc realloc
- #define static_strlen(string_literal) (sizeof(string_literal) - sizeof(""))
- #define can_read(buffer, size) ((buffer != NULL) && (((buffer)->offset + size) <= (buffer)->length))
- #define can_access_at_index(buffer, index) ((buffer != NULL) && (((buffer)->offset + index) < (buffer)->length))
- #define cannot_access_at_index(buffer, index) (!can_access_at_index(buffer, index))
- #define buffer_at_offset(buffer) ((buffer)->content + (buffer)->offset)
- #define cjson_min(a, b) (((a) < (b)) ? (a) : (b))

Typedefs

- typedef struct internal_hooks internal_hooks

Functions

- CJSON_PUBLIC (const char *)
- CJSON_PUBLIC (char *)
- CJSON_PUBLIC (double)
- CJSON_PUBLIC (void)
- CJSON_PUBLIC (cJSON *)
- CJSON_PUBLIC (cJSON_bool)
- cJSON * cJSON_Duplicate_rec (const cJSON *item, size_t depth, cJSON_bool recurse)
- CJSON_PUBLIC (void *)

4.7.1 Macro Definition Documentation

4.7.1.1 buffer_at_offset

```
#define buffer_at_offset(
              buffer ) ((buffer)->content + (buffer)->offset)
```

Definition at line 272 of file cJSON.c.

4.7.1.2 can_access_at_index

```
#define can_access_at_index(
              buffer,
              index ) ((buffer != NULL) && (((buffer)->offset + index) < (buffer)->length))
```

Definition at line 269 of file cJSON.c.

4.7.1.3 can_read

```
#define can_read(
              buffer,
              size ) ((buffer != NULL) && (((buffer)->offset + size) <= (buffer)->length))
```

Definition at line 267 of file cJSON.c.

4.7.1.4 cannot_access_at_index

```
#define cannot_access_at_index(
              buffer,
              index ) (!can_access_at_index(buffer, index))
```

Definition at line 270 of file cJSON.c.

4.7.1.5 cjson_min

```
#define cjson_min(
              a,
              b ) (((a) < (b)) ?  (a) :  (b))
```

Definition at line 1031 of file cJSON.c.

4.7.1.6 false

```
#define false ((cJSON_bool)0)
```

Definition at line 70 of file cJSON.c.

4.7.1.7 internal_free

```
#define internal_free free
```

Definition at line 164 of file cJSON.c.

4.7.1.8 internal_malloc

```
#define internal_malloc malloc
```

Definition at line 163 of file cJSON.c.

4.7.1.9 internal_realloc

```
#define internal_realloc realloc
```

Definition at line 165 of file cJSON.c.

4.7.1.10 isinf

```
#define isinf(
              d ) (isnan((d - d)) && !isnan(d))
```

Definition at line 74 of file cJSON.c.

4.7.1.11 isnan

```
#define isnan(
              d ) (d != d)
```

Definition at line 77 of file cJSON.c.

4.7.1.12 NAN

```
#define NAN 0.0/0.0
```

Definition at line 84 of file cJSON.c.

4.7.1.13 static_strlen

```
#define static_strlen(
              string_literal ) (sizeof(string_literal) - sizeof(""))
```

Definition at line 169 of file cJSON.c.

4.7.1.14 true

```
#define true ((cJSON_bool)1)
```

Definition at line 65 of file cJSON.c.

4.7.2 Typedef Documentation

4.7.2.1 internal_hooks

```
typedef struct internal_hooks internal_hooks
```

4.7.3 Function Documentation

4.7.3.1 cJSON_Duplicate_rec()

```
cJSON * cJSON_Duplicate_rec (
              const cJSON * item,
              size_t depth,
              cJSON_bool recurse )
```

Definition at line 2325 of file cJSON.c.

4.7.3.2 CJSON_PUBLIC() [1/7]

```
CJSON_PUBLIC (
              char *  )
```

Definition at line 98 of file cJSON.c.

4.7.3.3 CJSON_PUBLIC() [2/7]

```
CJSON_PUBLIC (
             cJSON *  )
```

Definition at line 942 of file cJSON.c.

4.7.3.4 CJSON_PUBLIC() [3/7]

```
CJSON_PUBLIC (
             cJSON_bool  )
```

Definition at line 1127 of file cJSON.c.

4.7.3.5 CJSON_PUBLIC() [4/7]

```
CJSON_PUBLIC (
             const char *  )
```

Definition at line 94 of file cJSON.c.

4.7.3.6 CJSON_PUBLIC() [5/7]

```
CJSON_PUBLIC (
             double  )
```

Definition at line 106 of file cJSON.c.

4.7.3.7 CJSON_PUBLIC() [6/7]

```
CJSON_PUBLIC (
             void *  )
```

Definition at line 2668 of file cJSON.c.

4.7.3.8 CJSON_PUBLIC() [7/7]

```
CJSON_PUBLIC (
             void  )
```

Definition at line 191 of file cJSON.c.

4.8 cJSON.c

Go to the documentation of this file.

```
00001 /*
00002 Copyright (c) 2009-2017 Dave Gamble and cJSON contributors
00003
00004 Permission is hereby granted, free of charge, to any person obtaining a copy
00005 of this software and associated documentation files (the "Software"), to deal
00006 in the Software without restriction, including without limitation the rights
00007 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00008 copies of the Software, and to permit persons to whom the Software is
00009 furnished to do so, subject to the following conditions:
00010
00011 The above copyright notice and this permission notice shall be included in
00012 all copies or substantial portions of the Software.
00013
00014 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00015 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00016 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00017 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00018 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00019 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
00020 THE SOFTWARE.
00021 */
00022
00023 /* cJSON */
00024 /* JSON parser in C. */
00025
00026 /* disable warnings about old C89 functions in MSVC */
00027 #if !defined(_CRT_SECURE_NO_DEPRECATE) && defined(_MSC_VER)
00028 #define _CRT_SECURE_NO_DEPRECATE
00029 #endif
00030
00031 #ifdef __GNUC__
00032 #pragma GCC visibility push(default)
00033 #endif
00034 #if defined(_MSC_VER)
00035 #pragma warning (push)
00036 /* disable warning about single line comments in system headers */
00037 #pragma warning (disable :  4001)
00038 #endif
00039
00040 #include <string.h>
00041 #include <stdio.h>
00042 #include <math.h>
00043 #include <stdlib.h>
00044 #include <limits.h>
00045 #include <ctype.h>
00046 #include <float.h>
00047
00048 #ifdef ENABLE_LOCALES
00049 #include <locale.h>
00050 #endif
00051
00052 #if defined(_MSC_VER)
00053 #pragma warning (pop)
00054 #endif
00055 #ifdef __GNUC__
00056 #pragma GCC visibility pop
00057 #endif
00058
00059 #include "cJSON.h"
00060
00061 /* define our own boolean type */
00062 #ifdef true
00063 #undef true
00064 #endif
00065 #define true ((cJSON_bool)1)
00066
00067 #ifdef false
00068 #undef false
00069 #endif
00070 #define false ((cJSON_bool)0)
00071
00072 /* define isnan and isinf for ANSI C, if in C99 or above, isnan and isinf has been defined in math.h
      */
00073 #ifndef isinf
00074 #define isinf(d) (isnan((d - d)) && !isnan(d))
00075 #endif
00076 #ifndef isnan
00077 #define isnan(d) (d != d)
00078 #endif
00079
00080 #ifndef NAN
00081 #ifdef _WIN32
```

```
00082 #define NAN sqrt(-1.0)
00083 #else
00084 #define NAN 0.0/0.0
00085 #endif
00086 #endif
00087
00088 typedef struct {
00089     const unsigned char *json;
00090     size_t position;
00091 } error;
00092 static error global_error = { NULL, 0 };
00093
00094 CJSON_PUBLIC(const char *) cJSON_GetErrorPtr(void) {
00095     return (const char*) (global_error.json + global_error.position);
00096 }
00097
00098 CJSON_PUBLIC(char *) cJSON_GetStringValue(const cJSON * const item) {
00099     if (!cJSON_IsString(item)) {
00100         return NULL;
00101     }
00102
00103     return item->valuestring;
00104 }
00105
00106 CJSON_PUBLIC(double) cJSON_GetNumberValue(const cJSON * const item) {
00107     if (!cJSON_IsNumber(item)) {
00108         return (double) NAN;
00109     }
00110
00111     return item->valuedouble;
00112 }
00113
00114 /* This is a safeguard to prevent copy-pasters from using incompatible C and header files */
00115 #if (CJSON_VERSION_MAJOR != 1) || (CJSON_VERSION_MINOR != 7) || (CJSON_VERSION_PATCH != 18)
00116 #error cJSON.h and cJSON.c have different versions.  Make sure that both have the same.
00117 #endif
00118
00119 CJSON_PUBLIC(const char*) cJSON_Version(void) {
00120     static char version[15];
00121     sprintf(version, "%i.%i.%i", CJSON_VERSION_MAJOR, CJSON_VERSION_MINOR, CJSON_VERSION_PATCH);
00122
00123     return version;
00124 }
00125
00126 /* Case insensitive string comparison, doesn't consider two NULL pointers equal though */
00127 static int case_insensitive_strcmp(const unsigned char *string1, const unsigned char *string2) {
00128     if ((string1 == NULL) || (string2 == NULL)) {
00129         return 1;
00130     }
00131
00132     if (string1 == string2) {
00133         return 0;
00134     }
00135
00136     for(; tolower(*string1) == tolower(*string2); (void)string1++, string2++) {
00137         if (*string1 == '\0') {
00138             return 0;
00139         }
00140     }
00141
00142     return tolower(*string1) - tolower(*string2);
00143 }
00144
00145 typedef struct internal_hooks {
00146     void *(CJSON_CDECL *allocate)(size_t size);
00147     void (CJSON_CDECL *deallocate)(void *pointer);
00148     void *(CJSON_CDECL *reallocate)(void *pointer, size_t size);
00149 } internal_hooks;
00150
00151 #if defined(_MSC_VER)
00152 /* work around MSVC error C2322:  '...'  address of dllimport '...'  is not static */
00153 static void * CJSON_CDECL internal_malloc(size_t size) {
00154     return malloc(size);
00155 }
00156 static void CJSON_CDECL internal_free(void *pointer) {
00157     free(pointer);
00158 }
00159 static void * CJSON_CDECL internal_realloc(void *pointer, size_t size) {
00160     return realloc(pointer, size);
00161 }
00162 #else
00163 #define internal_malloc malloc
00164 #define internal_free free
00165 #define internal_realloc realloc
00166 #endif
00167
00168 /* strlen of character literals resolved at compile time */
```

```
00169 #define static_strlen(string_literal) (sizeof(string_literal) - sizeof(""))
00170
00171 static internal_hooks global_hooks = { internal_malloc, internal_free, internal_realloc };
00172
00173 static unsigned char* cJSON_strdup(const unsigned char* string, const internal_hooks * const hooks) {
00174     size_t length = 0;
00175     unsigned char *copy = NULL;
00176
00177     if (string == NULL) {
00178         return NULL;
00179     }
00180
00181     length = strlen((const char*)string) + sizeof("");
00182     copy = (unsigned char*)hooks->allocate(length);
00183     if (copy == NULL) {
00184         return NULL;
00185     }
00186     memcpy(copy, string, length);
00187
00188     return copy;
00189 }
00190
00191 CJSON_PUBLIC(void) cJSON_InitHooks(cJSON_Hooks* hooks) {
00192     if (hooks == NULL) {
00193         /* Reset hooks */
00194         global_hooks.allocate = malloc;
00195         global_hooks.deallocate = free;
00196         global_hooks.reallocate = realloc;
00197         return;
00198     }
00199
00200     global_hooks.allocate = malloc;
00201     if (hooks->malloc_fn != NULL) {
00202         global_hooks.allocate = hooks->malloc_fn;
00203     }
00204
00205     global_hooks.deallocate = free;
00206     if (hooks->free_fn != NULL) {
00207         global_hooks.deallocate = hooks->free_fn;
00208     }
00209
00210     /* use realloc only if both free and malloc are used */
00211     global_hooks.reallocate = NULL;
00212     if ((global_hooks.allocate == malloc) && (global_hooks.deallocate == free)) {
00213         global_hooks.reallocate = realloc;
00214     }
00215 }
00216
00217 /* Internal constructor.  */
00218 static cJSON *cJSON_New_Item(const internal_hooks * const hooks) {
00219     cJSON* node = (cJSON*)hooks->allocate(sizeof(cJSON));
00220     if (node) {
00221         memset(node, '\0', sizeof(cJSON));
00222     }
00223
00224     return node;
00225 }
00226
00227 /* Delete a cJSON structure.  */
00228 CJSON_PUBLIC(void) cJSON_Delete(cJSON *item) {
00229     cJSON *next = NULL;
00230     while (item != NULL) {
00231         next = item->next;
00232         if (!(item->type & cJSON_IsReference) && (item->child != NULL)) {
00233             cJSON_Delete(item->child);
00234         }
00235         if (!(item->type & cJSON_IsReference) && (item->valuestring != NULL)) {
00236             global_hooks.deallocate(item->valuestring);
00237             item->valuestring = NULL;
00238         }
00239         if (!(item->type & cJSON_StringIsConst) && (item->string != NULL)) {
00240             global_hooks.deallocate(item->string);
00241             item->string = NULL;
00242         }
00243         global_hooks.deallocate(item);
00244         item = next;
00245     }
00246 }
00247
00248 /* get the decimal point character of the current locale */
00249 static unsigned char get_decimal_point(void) {
00250 #ifdef ENABLE_LOCALES
00251     struct lconv *lconv = localeconv();
00252     return (unsigned char) lconv->decimal_point[0];
00253 #else
00254     return '.';
00255 #endif
```

```
00256 }
00257
00258 typedef struct {
00259     const unsigned char *content;
00260     size_t length;
00261     size_t offset;
00262     size_t depth; /* How deeply nested (in arrays/objects) is the input at the current offset.  */
00263     internal_hooks hooks;
00264 } parse_buffer;
00265
00266 /* check if the given size is left to read in a given parse buffer (starting with 1) */
00267 #define can_read(buffer, size) ((buffer != NULL) && (((buffer)->offset + size) <= (buffer)->length))
00268 /* check if the buffer can be accessed at the given index (starting with 0) */
00269 #define can_access_at_index(buffer, index) ((buffer != NULL) && (((buffer)->offset + index) <
      (buffer)->length))
00270 #define cannot_access_at_index(buffer, index) (!can_access_at_index(buffer, index))
00271 /* get a pointer to the buffer at the position */
00272 #define buffer_at_offset(buffer) ((buffer)->content + (buffer)->offset)
00273
00274 /* Parse the input text to generate a number, and populate the result into item.  */
00275 static cJSON_bool parse_number(cJSON * const item, parse_buffer * const input_buffer) {
00276     double number = 0;
00277     unsigned char *after_end = NULL;
00278     unsigned char number_c_string[64];
00279     unsigned char decimal_point = get_decimal_point();
00280     size_t i = 0;
00281
00282     if ((input_buffer == NULL) || (input_buffer->content == NULL)) {
00283         return false;
00284     }
00285
00286     /* copy the number into a temporary buffer and replace '.'  with the decimal point
00287 * of the current locale (for strtod)
00288 * This also takes care of '\0' not necessarily being available for marking the end of the input */
00289     for (i = 0; (i < (sizeof(number_c_string) - 1)) && can_access_at_index(input_buffer, i); i++) {
00290         switch (buffer_at_offset(input_buffer)[i]) {
00291         case '0':
00292         case '1':
00293         case '2':
00294         case '3':
00295         case '4':
00296         case '5':
00297         case '6':
00298         case '7':
00299         case '8':
00300         case '9':
00301         case '+':
00302         case '-':
00303         case 'e':
00304         case 'E':
00305             number_c_string[i] = buffer_at_offset(input_buffer)[i];
00306             break;
00307
00308         case '.':
00309             number_c_string[i] = decimal_point;
00310             break;
00311
00312         default:
00313             goto loop_end;
00314         }
00315     }
00316 loop_end:
00317     number_c_string[i] = '\0';
00318
00319     number = strtod((const char*)number_c_string, (char**)&after_end);
00320     if (number_c_string == after_end) {
00321         return false; /* parse_error */
00322     }
00323
00324     item->valuedouble = number;
00325
00326     /* use saturation in case of overflow */
00327     if (number >= INT_MAX) {
00328         item->valueint = INT_MAX;
00329     } else if (number <= (double)INT_MIN) {
00330         item->valueint = INT_MIN;
00331     } else {
00332         item->valueint = (int)number;
00333     }
00334
00335     item->type = cJSON_Number;
00336
00337     input_buffer->offset += (size_t)(after_end - number_c_string);
00338     return true;
00339 }
00340
00341 /* don't ask me, but the original cJSON_SetNumberValue returns an integer or double */
```

```
00342 CJSON_PUBLIC(double) cJSON_SetNumberHelper(cJSON *object, double number) {
00343     if (number >= INT_MAX) {
00344         object->valueint = INT_MAX;
00345     } else if (number <= (double)INT_MIN) {
00346         object->valueint = INT_MIN;
00347     } else {
00348         object->valueint = (int)number;
00349     }
00350
00351     return object->valuedouble = number;
00352 }
00353
00354 /* Note:  when passing a NULL valuestring, cJSON_SetValuestring treats this as an error and return
      NULL */
00355 CJSON_PUBLIC(char*) cJSON_SetValuestring(cJSON *object, const char *valuestring) {
00356     char *copy = NULL;
00357     size_t v1_len;
00358     size_t v2_len;
00359     /* if object's type is not cJSON_String or is cJSON_IsReference, it should not set valuestring */
00360     if ((object == NULL) || !(object->type & cJSON_String) || (object->type & cJSON_IsReference)) {
00361         return NULL;
00362     }
00363     /* return NULL if the object is corrupted or valuestring is NULL */
00364     if (object->valuestring == NULL || valuestring == NULL) {
00365         return NULL;
00366     }
00367
00368     v1_len = strlen(valuestring);
00369     v2_len = strlen(object->valuestring);
00370
00371     if (v1_len <= v2_len) {
00372         /* strcpy does not handle overlapping string:  [X1, X2] [Y1, Y2] => X2 < Y1 or Y2 < X1 */
00373         if (!( valuestring + v1_len < object->valuestring || object->valuestring + v2_len <
      valuestring )) {
00374             return NULL;
00375         }
00376         strcpy(object->valuestring, valuestring);
00377         return object->valuestring;
00378     }
00379     copy = (char*) cJSON_strdup((const unsigned char*)valuestring, &global_hooks);
00380     if (copy == NULL) {
00381         return NULL;
00382     }
00383     if (object->valuestring != NULL) {
00384         cJSON_free(object->valuestring);
00385     }
00386     object->valuestring = copy;
00387
00388     return copy;
00389 }
00390
00391 typedef struct {
00392     unsigned char *buffer;
00393     size_t length;
00394     size_t offset;
00395     size_t depth; /* current nesting depth (for formatted printing) */
00396     cJSON_bool noalloc;
00397     cJSON_bool format; /* is this print a formatted print */
00398     internal_hooks hooks;
00399 } printbuffer;
00400
00401 /* realloc printbuffer if necessary to have at least "needed" bytes more */
00402 static unsigned char* ensure(printbuffer * const p, size_t needed) {
00403     unsigned char *newbuffer = NULL;
00404     size_t newsize = 0;
00405
00406     if ((p == NULL) || (p->buffer == NULL)) {
00407         return NULL;
00408     }
00409
00410     if ((p->length > 0) && (p->offset >= p->length)) {
00411         /* make sure that offset is valid */
00412         return NULL;
00413     }
00414
00415     if (needed > INT_MAX) {
00416         /* sizes bigger than INT_MAX are currently not supported */
00417         return NULL;
00418     }
00419
00420     needed += p->offset + 1;
00421     if (needed <= p->length) {
00422         return p->buffer + p->offset;
00423     }
00424
00425     if (p->noalloc) {
00426         return NULL;
```

```
00427     }
00428
00429     /* calculate new buffer size */
00430     if (needed > (INT_MAX / 2)) {
00431         /* overflow of int, use INT_MAX if possible */
00432         if (needed <= INT_MAX) {
00433             newsize = INT_MAX;
00434         } else {
00435             return NULL;
00436         }
00437     } else {
00438         newsize = needed * 2;
00439     }
00440
00441     if (p->hooks.reallocate != NULL) {
00442         /* reallocate with realloc if available */
00443         newbuffer = (unsigned char*)p->hooks.reallocate(p->buffer, newsize);
00444         if (newbuffer == NULL) {
00445             p->hooks.deallocate(p->buffer);
00446             p->length = 0;
00447             p->buffer = NULL;
00448
00449             return NULL;
00450         }
00451     } else {
00452         /* otherwise reallocate manually */
00453         newbuffer = (unsigned char*)p->hooks.allocate(newsize);
00454         if (!newbuffer) {
00455             p->hooks.deallocate(p->buffer);
00456             p->length = 0;
00457             p->buffer = NULL;
00458
00459             return NULL;
00460         }
00461
00462         memcpy(newbuffer, p->buffer, p->offset + 1);
00463         p->hooks.deallocate(p->buffer);
00464     }
00465     p->length = newsize;
00466     p->buffer = newbuffer;
00467
00468     return newbuffer + p->offset;
00469 }
00470
00471 /* calculate the new length of the string in a printbuffer and update the offset */
00472 static void update_offset(printbuffer * const buffer) {
00473     const unsigned char *buffer_pointer = NULL;
00474     if ((buffer == NULL) || (buffer->buffer == NULL)) {
00475         return;
00476     }
00477     buffer_pointer = buffer->buffer + buffer->offset;
00478
00479     buffer->offset += strlen((const char*)buffer_pointer);
00480 }
00481
00482 /* securely comparison of floating-point variables */
00483 static cJSON_bool compare_double(double a, double b) {
00484     double maxVal = fabs(a) > fabs(b) ?  fabs(a) :  fabs(b);
00485     return (fabs(a - b) <= maxVal * DBL_EPSILON);
00486 }
00487
00488 /* Render the number nicely from the given item into a string.  */
00489 static cJSON_bool print_number(const cJSON * const item, printbuffer * const output_buffer) {
00490     unsigned char *output_pointer = NULL;
00491     double d = item->valuedouble;
00492     int length = 0;
00493     size_t i = 0;
00494     unsigned char number_buffer[26] = {0}; /* temporary buffer to print the number into */
00495     unsigned char decimal_point = get_decimal_point();
00496     double test = 0.0;
00497
00498     if (output_buffer == NULL) {
00499         return false;
00500     }
00501
00502     /* This checks for NaN and Infinity */
00503     if (isnan(d) || isinf(d)) {
00504         length = sprintf((char*)number_buffer, "null");
00505     } else if(d == (double)item->valueint) {
00506         length = sprintf((char*)number_buffer, "%d", item->valueint);
00507     } else {
00508         /* Try 15 decimal places of precision to avoid nonsignificant nonzero digits */
00509         length = sprintf((char*)number_buffer, "%1.15g", d);
00510
00511         /* Check whether the original double can be recovered */
00512         if ((sscanf((char*)number_buffer, "%lg", &test) != 1) || !compare_double((double)test, d)) {
00513             /* If not, print with 17 decimal places of precision */
```

```
00514             length = sprintf((char*)number_buffer, "%1.17g", d);
00515         }
00516     }
00517
00518     /* sprintf failed or buffer overrun occurred */
00519     if ((length < 0) || (length > (int)(sizeof(number_buffer) - 1))) {
00520         return false;
00521     }
00522
00523     /* reserve appropriate space in the output */
00524     output_pointer = ensure(output_buffer, (size_t)length + sizeof(""));
00525     if (output_pointer == NULL) {
00526         return false;
00527     }
00528
00529     /* copy the printed number to the output and replace locale
00530 * dependent decimal point with '.'  */
00531     for (i = 0; i < ((size_t)length); i++) {
00532         if (number_buffer[i] == decimal_point) {
00533             output_pointer[i] = '.';
00534             continue;
00535         }
00536
00537         output_pointer[i] = number_buffer[i];
00538     }
00539     output_pointer[i] = '\0';
00540
00541     output_buffer->offset += (size_t)length;
00542
00543     return true;
00544 }
00545
00546 /* parse 4 digit hexadecimal number */
00547 static unsigned parse_hex4(const unsigned char * const input) {
00548     unsigned int h = 0;
00549     size_t i = 0;
00550
00551     for (i = 0; i < 4; i++) {
00552         /* parse digit */
00553         if ((input[i] >= '0') && (input[i] <= '9')) {
00554             h += (unsigned int) input[i] - '0';
00555         } else if ((input[i] >= 'A') && (input[i] <= 'F')) {
00556             h += (unsigned int) 10 + input[i] - 'A';
00557         } else if ((input[i] >= 'a') && (input[i] <= 'f')) {
00558             h += (unsigned int) 10 + input[i] - 'a';
00559         } else { /* invalid */
00560             return 0;
00561         }
00562
00563         if (i < 3) {
00564             /* shift left to make place for the next nibble */
00565             h = h « 4;
00566         }
00567     }
00568
00569     return h;
00570 }
00571
00572 /* converts a UTF-16 literal to UTF-8
00573 * A literal can be one or two sequences of the form \uXXXX */
00574 static unsigned char utf16_literal_to_utf8(const unsigned char * const input_pointer, const unsigned
      char * const input_end, unsigned char **output_pointer) {
00575     long unsigned int codepoint = 0;
00576     unsigned int first_code = 0;
00577     const unsigned char *first_sequence = input_pointer;
00578     unsigned char utf8_length = 0;
00579     unsigned char utf8_position = 0;
00580     unsigned char sequence_length = 0;
00581     unsigned char first_byte_mark = 0;
00582
00583     if ((input_end - first_sequence) < 6) {
00584         /* input ends unexpectedly */
00585         goto fail;
00586     }
00587
00588     /* get the first utf16 sequence */
00589     first_code = parse_hex4(first_sequence + 2);
00590
00591     /* check that the code is valid */
00592     if (((first_code >= 0xDC00) && (first_code <= 0xDFFF))) {
00593         goto fail;
00594     }
00595
00596     /* UTF16 surrogate pair */
00597     if ((first_code >= 0xD800) && (first_code <= 0xDBFF)) {
00598         const unsigned char *second_sequence = first_sequence + 6;
00599         unsigned int second_code = 0;
```

```
00600         sequence_length = 12; /* \uXXXX\uXXXX */
00601
00602         if ((input_end - second_sequence) < 6) {
00603             /* input ends unexpectedly */
00604             goto fail;
00605         }
00606
00607         if ((second_sequence[0] != '\\') || (second_sequence[1] != 'u')) {
00608             /* missing second half of the surrogate pair */
00609             goto fail;
00610         }
00611
00612         /* get the second utf16 sequence */
00613         second_code = parse_hex4(second_sequence + 2);
00614         /* check that the code is valid */
00615         if ((second_code < 0xDC00) || (second_code > 0xDFFF)) {
00616             /* invalid second half of the surrogate pair */
00617             goto fail;
00618         }
00619
00620
00621         /* calculate the unicode codepoint from the surrogate pair */
00622         codepoint = 0x10000 + (((first_code & 0x3FF) << 10) | (second_code & 0x3FF));
00623     } else {
00624         sequence_length = 6; /* \uXXXX */
00625         codepoint = first_code;
00626     }
00627
00628     /* encode as UTF-8
00629  * takes at maximum 4 bytes to encode:
00630  * 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx */
00631     if (codepoint < 0x80) {
00632         /* normal ascii, encoding 0xxxxxxx */
00633         utf8_length = 1;
00634     } else if (codepoint < 0x800) {
00635         /* two bytes, encoding 110xxxxx 10xxxxxx */
00636         utf8_length = 2;
00637         first_byte_mark = 0xC0; /* 11000000 */
00638     } else if (codepoint < 0x10000) {
00639         /* three bytes, encoding 1110xxxx 10xxxxxx 10xxxxxx */
00640         utf8_length = 3;
00641         first_byte_mark = 0xE0; /* 11100000 */
00642     } else if (codepoint <= 0x10FFFF) {
00643         /* four bytes, encoding 1110xxxx 10xxxxxx 10xxxxxx 10xxxxxx */
00644         utf8_length = 4;
00645         first_byte_mark = 0xF0; /* 11110000 */
00646     } else {
00647         /* invalid unicode codepoint */
00648         goto fail;
00649     }
00650
00651     /* encode as utf8 */
00652     for (utf8_position = (unsigned char)(utf8_length - 1); utf8_position > 0; utf8_position--) {
00653         /* 10xxxxxx */
00654         (*output_pointer)[utf8_position] = (unsigned char)((codepoint | 0x80) & 0xBF);
00655         codepoint >>= 6;
00656     }
00657     /* encode first byte */
00658     if (utf8_length > 1) {
00659         (*output_pointer)[0] = (unsigned char)((codepoint | first_byte_mark) & 0xFF);
00660     } else {
00661         (*output_pointer)[0] = (unsigned char)(codepoint & 0x7F);
00662     }
00663
00664     *output_pointer += utf8_length;
00665
00666     return sequence_length;
00667
00668 fail:
00669     return 0;
00670 }
00671
00672 /* Parse the input text into an unescaped cinput, and populate item.  */
00673 static cJSON_bool parse_string(cJSON * const item, parse_buffer * const input_buffer) {
00674     const unsigned char *input_pointer = buffer_at_offset(input_buffer) + 1;
00675     const unsigned char *input_end = buffer_at_offset(input_buffer) + 1;
00676     unsigned char *output_pointer = NULL;
00677     unsigned char *output = NULL;
00678
00679     /* not a string */
00680     if (buffer_at_offset(input_buffer)[0] != '\"') {
00681         goto fail;
00682     }
00683
00684     {
00685         /* calculate approximate size of the output (overestimate) */
00686         size_t allocation_length = 0;
```

```
00687         size_t skipped_bytes = 0;
00688         while (((size_t)(input_end - input_buffer->content) < input_buffer->length) && (*input_end !=
    '\"')) {
00689             /* is escape sequence */
00690             if (input_end[0] == '\\') {
00691                 if ((size_t)(input_end + 1 - input_buffer->content) >= input_buffer->length) {
00692                     /* prevent buffer overflow when last input character is a backslash */
00693                     goto fail;
00694                 }
00695                 skipped_bytes++;
00696                 input_end++;
00697             }
00698             input_end++;
00699         }
00700         if (((size_t)(input_end - input_buffer->content) >= input_buffer->length) || (*input_end !=
    '\"')) {
00701             goto fail; /* string ended unexpectedly */
00702         }
00703 
00704         /* This is at most how much we need for the output */
00705         allocation_length = (size_t) (input_end - buffer_at_offset(input_buffer)) - skipped_bytes;
00706         output = (unsigned char*)input_buffer->hooks.allocate(allocation_length + sizeof(""));
00707         if (output == NULL) {
00708             goto fail; /* allocation failure */
00709         }
00710     }
00711 
00712     output_pointer = output;
00713     /* loop through the string literal */
00714     while (input_pointer < input_end) {
00715         if (*input_pointer != '\\') {
00716             *output_pointer++ = *input_pointer++;
00717         }
00718         /* escape sequence */
00719         else {
00720             unsigned char sequence_length = 2;
00721             if ((input_end - input_pointer) < 1) {
00722                 goto fail;
00723             }
00724 
00725             switch (input_pointer[1]) {
00726             case 'b':
00727                 *output_pointer++ = '\b';
00728                 break;
00729             case 'f':
00730                 *output_pointer++ = '\f';
00731                 break;
00732             case 'n':
00733                 *output_pointer++ = '\n';
00734                 break;
00735             case 'r':
00736                 *output_pointer++ = '\r';
00737                 break;
00738             case 't':
00739                 *output_pointer++ = '\t';
00740                 break;
00741             case '\"':
00742             case '\\':
00743             case '/':
00744                 *output_pointer++ = input_pointer[1];
00745                 break;
00746 
00747             /* UTF-16 literal */
00748             case 'u':
00749                 sequence_length = utf16_literal_to_utf8(input_pointer, input_end, &output_pointer);
00750                 if (sequence_length == 0) {
00751                     /* failed to convert UTF16-literal to UTF-8 */
00752                     goto fail;
00753                 }
00754                 break;
00755 
00756             default:
00757                 goto fail;
00758             }
00759             input_pointer += sequence_length;
00760         }
00761     }
00762 
00763     /* zero terminate the output */
00764     *output_pointer = '\0';
00765 
00766     item->type = cJSON_String;
00767     item->valuestring = (char*)output;
00768 
00769     input_buffer->offset = (size_t) (input_end - input_buffer->content);
00770     input_buffer->offset++;
00771 
```

```
00772     return true;
00773
00774 fail:
00775     if (output != NULL) {
00776         input_buffer->hooks.deallocate(output);
00777         output = NULL;
00778     }
00779
00780     if (input_pointer != NULL) {
00781         input_buffer->offset = (size_t)(input_pointer - input_buffer->content);
00782     }
00783
00784     return false;
00785 }
00786
00787 /* Render the cstring provided to an escaped version that can be printed.  */
00788 static cJSON_bool print_string_ptr(const unsigned char * const input, printbuffer * const
      output_buffer) {
00789     const unsigned char *input_pointer = NULL;
00790     unsigned char *output = NULL;
00791     unsigned char *output_pointer = NULL;
00792     size_t output_length = 0;
00793     /* numbers of additional characters needed for escaping */
00794     size_t escape_characters = 0;
00795
00796     if (output_buffer == NULL) {
00797         return false;
00798     }
00799
00800     /* empty string */
00801     if (input == NULL) {
00802         output = ensure(output_buffer, sizeof("\"\""));
00803         if (output == NULL) {
00804             return false;
00805         }
00806         strcpy((char*)output, "\"\"");
00807
00808         return true;
00809     }
00810
00811     /* set "flag" to 1 if something needs to be escaped */
00812     for (input_pointer = input; *input_pointer; input_pointer++) {
00813         switch (*input_pointer) {
00814         case '\"':
00815         case '\\':
00816         case '\b':
00817         case '\f':
00818         case '\n':
00819         case '\r':
00820         case '\t':
00821             /* one character escape sequence */
00822             escape_characters++;
00823             break;
00824         default:
00825             if (*input_pointer < 32) {
00826                 /* UTF-16 escape sequence uXXXX */
00827                 escape_characters += 5;
00828             }
00829             break;
00830         }
00831     }
00832     output_length = (size_t)(input_pointer - input) + escape_characters;
00833
00834     output = ensure(output_buffer, output_length + sizeof("\"\""));
00835     if (output == NULL) {
00836         return false;
00837     }
00838
00839     /* no characters have to be escaped */
00840     if (escape_characters == 0) {
00841         output[0] = '\"';
00842         memcpy(output + 1, input, output_length);
00843         output[output_length + 1] = '\"';
00844         output[output_length + 2] = '\0';
00845
00846         return true;
00847     }
00848
00849     output[0] = '\"';
00850     output_pointer = output + 1;
00851     /* copy the string */
00852     for (input_pointer = input; *input_pointer != '\0'; (void)input_pointer++, output_pointer++) {
00853         if ((*input_pointer > 31) && (*input_pointer != '\"') && (*input_pointer != '\\')) {
00854             /* normal character, copy */
00855             *output_pointer = *input_pointer;
00856         } else {
00857             /* character needs to be escaped */
```

```
00858             *output_pointer++ = '\\';
00859             switch (*input_pointer) {
00860             case '\\':
00861                 *output_pointer = '\\';
00862                 break;
00863             case '\"':
00864                 *output_pointer = '\"';
00865                 break;
00866             case '\b':
00867                 *output_pointer = 'b';
00868                 break;
00869             case '\f':
00870                 *output_pointer = 'f';
00871                 break;
00872             case '\n':
00873                 *output_pointer = 'n';
00874                 break;
00875             case '\r':
00876                 *output_pointer = 'r';
00877                 break;
00878             case '\t':
00879                 *output_pointer = 't';
00880                 break;
00881             default:
00882                 /* escape and print as unicode codepoint */
00883                 sprintf((char*)output_pointer, "u%04x", *input_pointer);
00884                 output_pointer += 4;
00885                 break;
00886             }
00887         }
00888     }
00889     output[output_length + 1] = '\"';
00890     output[output_length + 2] = '\0';
00891
00892     return true;
00893 }
00894
00895 /* Invoke print_string_ptr (which is useful) on an item.  */
00896 static cJSON_bool print_string(const cJSON * const item, printbuffer * const p) {
00897     return print_string_ptr((unsigned char*)item->valuestring, p);
00898 }
00899
00900 /* Predeclare these prototypes.  */
00901 static cJSON_bool parse_value(cJSON * const item, parse_buffer * const input_buffer);
00902 static cJSON_bool print_value(const cJSON * const item, printbuffer * const output_buffer);
00903 static cJSON_bool parse_array(cJSON * const item, parse_buffer * const input_buffer);
00904 static cJSON_bool print_array(const cJSON * const item, printbuffer * const output_buffer);
00905 static cJSON_bool parse_object(cJSON * const item, parse_buffer * const input_buffer);
00906 static cJSON_bool print_object(const cJSON * const item, printbuffer * const output_buffer);
00907
00908 /* Utility to jump whitespace and cr/lf */
00909 static parse_buffer *buffer_skip_whitespace(parse_buffer * const buffer) {
00910     if ((buffer == NULL) || (buffer->content == NULL)) {
00911         return NULL;
00912     }
00913
00914     if (cannot_access_at_index(buffer, 0)) {
00915         return buffer;
00916     }
00917
00918     while (can_access_at_index(buffer, 0) && (buffer_at_offset(buffer)[0] <= 32)) {
00919         buffer->offset++;
00920     }
00921
00922     if (buffer->offset == buffer->length) {
00923         buffer->offset--;
00924     }
00925
00926     return buffer;
00927 }
00928
00929 /* skip the UTF-8 BOM (byte order mark) if it is at the beginning of a buffer */
00930 static parse_buffer *skip_utf8_bom(parse_buffer * const buffer) {
00931     if ((buffer == NULL) || (buffer->content == NULL) || (buffer->offset != 0)) {
00932         return NULL;
00933     }
00934
00935     if (can_access_at_index(buffer, 4) && (strncmp((const char*)buffer_at_offset(buffer),
    "\xEF\xBB\xBF", 3) == 0)) {
00936         buffer->offset += 3;
00937     }
00938
00939     return buffer;
00940 }
00941
00942 CJSON_PUBLIC(cJSON *) cJSON_ParseWithOpts(const char *value, const char **return_parse_end, cJSON_bool
    require_null_terminated) {
```

```
00943     size_t buffer_length;
00944
00945     if (NULL == value) {
00946         return NULL;
00947     }
00948
00949     /* Adding null character size due to require_null_terminated.  */
00950     buffer_length = strlen(value) + sizeof("");
00951
00952     return cJSON_ParseWithLengthOpts(value, buffer_length, return_parse_end, require_null_terminated);
00953 }
00954
00955 /* Parse an object - create a new root, and populate.  */
00956 CJSON_PUBLIC(cJSON *) cJSON_ParseWithLengthOpts(const char *value, size_t buffer_length, const char
      **return_parse_end, cJSON_bool require_null_terminated) {
00957     parse_buffer buffer = { 0, 0, 0, 0, { 0, 0, 0 } };
00958     cJSON *item = NULL;
00959
00960     /* reset error position */
00961     global_error.json = NULL;
00962     global_error.position = 0;
00963
00964     if (value == NULL || 0 == buffer_length) {
00965         goto fail;
00966     }
00967
00968     buffer.content = (const unsigned char*)value;
00969     buffer.length = buffer_length;
00970     buffer.offset = 0;
00971     buffer.hooks = global_hooks;
00972
00973     item = cJSON_New_Item(&global_hooks);
00974     if (item == NULL) { /* memory fail */
00975         goto fail;
00976     }
00977
00978     if (!parse_value(item, buffer_skip_whitespace(skip_utf8_bom(&buffer)))) {
00979         /* parse failure.  ep is set.  */
00980         goto fail;
00981     }
00982
00983     /* if we require null-terminated JSON without appended garbage, skip and then check for a null
      terminator */
00984     if (require_null_terminated) {
00985         buffer_skip_whitespace(&buffer);
00986         if ((buffer.offset >= buffer.length) || buffer_at_offset(&buffer)[0] != '\0') {
00987             goto fail;
00988         }
00989     }
00990     if (return_parse_end) {
00991         *return_parse_end = (const char*)buffer_at_offset(&buffer);
00992     }
00993
00994     return item;
00995
00996 fail:
00997     if (item != NULL) {
00998         cJSON_Delete(item);
00999     }
01000
01001     if (value != NULL) {
01002         error local_error;
01003         local_error.json = (const unsigned char*)value;
01004         local_error.position = 0;
01005
01006         if (buffer.offset < buffer.length) {
01007             local_error.position = buffer.offset;
01008         } else if (buffer.length > 0) {
01009             local_error.position = buffer.length - 1;
01010         }
01011
01012         if (return_parse_end != NULL) {
01013             *return_parse_end = (const char*)local_error.json + local_error.position;
01014         }
01015
01016         global_error = local_error;
01017     }
01018
01019     return NULL;
01020 }
01021
01022 /* Default options for cJSON_Parse */
01023 CJSON_PUBLIC(cJSON *) cJSON_Parse(const char *value) {
01024     return cJSON_ParseWithOpts(value, 0, 0);
01025 }
01026
01027 CJSON_PUBLIC(cJSON *) cJSON_ParseWithLength(const char *value, size_t buffer_length) {
```

```
01028     return cJSON_ParseWithLengthOpts(value, buffer_length, 0, 0);
01029 }
01030
01031 #define cjson_min(a, b) (((a) < (b)) ? (a) : (b))
01032
01033 static unsigned char *print(const cJSON * const item, cJSON_bool format, const internal_hooks * const
      hooks) {
01034     static const size_t default_buffer_size = 256;
01035     printbuffer buffer[1];
01036     unsigned char *printed = NULL;
01037
01038     memset(buffer, 0, sizeof(buffer));
01039
01040     /* create buffer */
01041     buffer->buffer = (unsigned char*) hooks->allocate(default_buffer_size);
01042     buffer->length = default_buffer_size;
01043     buffer->format = format;
01044     buffer->hooks = *hooks;
01045     if (buffer->buffer == NULL) {
01046         goto fail;
01047     }
01048
01049     /* print the value */
01050     if (!print_value(item, buffer)) {
01051         goto fail;
01052     }
01053     update_offset(buffer);
01054
01055     /* check if reallocate is available */
01056     if (hooks->reallocate != NULL) {
01057         printed = (unsigned char*) hooks->reallocate(buffer->buffer, buffer->offset + 1);
01058         if (printed == NULL) {
01059             goto fail;
01060         }
01061         buffer->buffer = NULL;
01062     } else { /* otherwise copy the JSON over to a new buffer */
01063         printed = (unsigned char*) hooks->allocate(buffer->offset + 1);
01064         if (printed == NULL) {
01065             goto fail;
01066         }
01067         memcpy(printed, buffer->buffer, cjson_min(buffer->length, buffer->offset + 1));
01068         printed[buffer->offset] = '\0'; /* just to be sure */
01069
01070         /* free the buffer */
01071         hooks->deallocate(buffer->buffer);
01072         buffer->buffer = NULL;
01073     }
01074
01075     return printed;
01076
01077 fail:
01078     if (buffer->buffer != NULL) {
01079         hooks->deallocate(buffer->buffer);
01080         buffer->buffer = NULL;
01081     }
01082
01083     if (printed != NULL) {
01084         hooks->deallocate(printed);
01085         printed = NULL;
01086     }
01087
01088     return NULL;
01089 }
01090
01091 /* Render a cJSON item/entity/structure to text.  */
01092 CJSON_PUBLIC(char *) cJSON_Print(const cJSON *item) {
01093     return (char*)print(item, true, &global_hooks);
01094 }
01095
01096 CJSON_PUBLIC(char *) cJSON_PrintUnformatted(const cJSON *item) {
01097     return (char*)print(item, false, &global_hooks);
01098 }
01099
01100 CJSON_PUBLIC(char *) cJSON_PrintBuffered(const cJSON *item, int prebuffer, cJSON_bool fmt) {
01101     printbuffer p = { 0, 0, 0, 0, 0, 0, { 0, 0, 0 } };
01102
01103     if (prebuffer < 0) {
01104         return NULL;
01105     }
01106
01107     p.buffer = (unsigned char*)global_hooks.allocate((size_t)prebuffer);
01108     if (!p.buffer) {
01109         return NULL;
01110     }
01111
01112     p.length = (size_t)prebuffer;
01113     p.offset = 0;
```

```
01114     p.noalloc = false;
01115     p.format = fmt;
01116     p.hooks = global_hooks;
01117 
01118     if (!print_value(item, &p)) {
01119         global_hooks.deallocate(p.buffer);
01120         p.buffer = NULL;
01121         return NULL;
01122     }
01123 
01124     return (char*)p.buffer;
01125 }
01126 
01127 CJSON_PUBLIC(cJSON_bool) cJSON_PrintPreallocated(cJSON *item, char *buffer, const int length, const
      cJSON_bool format) {
01128     printbuffer p = { 0, 0, 0, 0, 0, 0, { 0, 0, 0 } };
01129 
01130     if ((length < 0) || (buffer == NULL)) {
01131         return false;
01132     }
01133 
01134     p.buffer = (unsigned char*)buffer;
01135     p.length = (size_t)length;
01136     p.offset = 0;
01137     p.noalloc = true;
01138     p.format = format;
01139     p.hooks = global_hooks;
01140 
01141     return print_value(item, &p);
01142 }
01143 
01144 /* Parser core - when encountering text, process appropriately.  */
01145 static cJSON_bool parse_value(cJSON * const item, parse_buffer * const input_buffer) {
01146     if ((input_buffer == NULL) || (input_buffer->content == NULL)) {
01147         return false; /* no input */
01148     }
01149 
01150     /* parse the different types of values */
01151     /* null */
01152     if (can_read(input_buffer, 4) && (strncmp((const char*)buffer_at_offset(input_buffer), "null", 4)
      == 0)) {
01153         item->type = cJSON_NULL;
01154         input_buffer->offset += 4;
01155         return true;
01156     }
01157     /* false */
01158     if (can_read(input_buffer, 5) && (strncmp((const char*)buffer_at_offset(input_buffer), "false", 5)
      == 0)) {
01159         item->type = cJSON_False;
01160         input_buffer->offset += 5;
01161         return true;
01162     }
01163     /* true */
01164     if (can_read(input_buffer, 4) && (strncmp((const char*)buffer_at_offset(input_buffer), "true", 4)
      == 0)) {
01165         item->type = cJSON_True;
01166         item->valueint = 1;
01167         input_buffer->offset += 4;
01168         return true;
01169     }
01170     /* string */
01171     if (can_access_at_index(input_buffer, 0) && (buffer_at_offset(input_buffer)[0] == '\"')) {
01172         return parse_string(item, input_buffer);
01173     }
01174     /* number */
01175     if (can_access_at_index(input_buffer, 0) && ((buffer_at_offset(input_buffer)[0] == '-') ||
      ((buffer_at_offset(input_buffer)[0] >= '0') && (buffer_at_offset(input_buffer)[0] <= '9')))) {
01176         return parse_number(item, input_buffer);
01177     }
01178     /* array */
01179     if (can_access_at_index(input_buffer, 0) && (buffer_at_offset(input_buffer)[0] == '[')) {
01180         return parse_array(item, input_buffer);
01181     }
01182     /* object */
01183     if (can_access_at_index(input_buffer, 0) && (buffer_at_offset(input_buffer)[0] == '{')) {
01184         return parse_object(item, input_buffer);
01185     }
01186 
01187     return false;
01188 }
01189 
01190 /* Render a value to text.  */
01191 static cJSON_bool print_value(const cJSON * const item, printbuffer * const output_buffer) {
01192     unsigned char *output = NULL;
01193 
01194     if ((item == NULL) || (output_buffer == NULL)) {
01195         return false;
```

```
01196     }
01197
01198     switch ((item->type) & 0xFF) {
01199     case cJSON_NULL:
01200         output = ensure(output_buffer, 5);
01201         if (output == NULL) {
01202             return false;
01203         }
01204         strcpy((char*)output, "null");
01205         return true;
01206
01207     case cJSON_False:
01208         output = ensure(output_buffer, 6);
01209         if (output == NULL) {
01210             return false;
01211         }
01212         strcpy((char*)output, "false");
01213         return true;
01214
01215     case cJSON_True:
01216         output = ensure(output_buffer, 5);
01217         if (output == NULL) {
01218             return false;
01219         }
01220         strcpy((char*)output, "true");
01221         return true;
01222
01223     case cJSON_Number:
01224         return print_number(item, output_buffer);
01225
01226     case cJSON_Raw:  {
01227         size_t raw_length = 0;
01228         if (item->valuestring == NULL) {
01229             return false;
01230         }
01231
01232         raw_length = strlen(item->valuestring) + sizeof("");
01233         output = ensure(output_buffer, raw_length);
01234         if (output == NULL) {
01235             return false;
01236         }
01237         memcpy(output, item->valuestring, raw_length);
01238         return true;
01239     }
01240
01241     case cJSON_String:
01242         return print_string(item, output_buffer);
01243
01244     case cJSON_Array:
01245         return print_array(item, output_buffer);
01246
01247     case cJSON_Object:
01248         return print_object(item, output_buffer);
01249
01250     default:
01251         return false;
01252     }
01253 }
01254
01255 /* Build an array from input text.  */
01256 static cJSON_bool parse_array(cJSON * const item, parse_buffer * const input_buffer) {
01257     cJSON *head = NULL; /* head of the linked list */
01258     cJSON *current_item = NULL;
01259
01260     if (input_buffer->depth >= CJSON_NESTING_LIMIT) {
01261         return false; /* to deeply nested */
01262     }
01263     input_buffer->depth++;
01264
01265     if (buffer_at_offset(input_buffer)[0] != '[') {
01266         /* not an array */
01267         goto fail;
01268     }
01269
01270     input_buffer->offset++;
01271     buffer_skip_whitespace(input_buffer);
01272     if (can_access_at_index(input_buffer, 0) && (buffer_at_offset(input_buffer)[0] == ']')) {
01273         /* empty array */
01274         goto success;
01275     }
01276
01277     /* check if we skipped to the end of the buffer */
01278     if (cannot_access_at_index(input_buffer, 0)) {
01279         input_buffer->offset--;
01280         goto fail;
01281     }
01282
```

```
01283     /* step back to character in front of the first element */
01284     input_buffer->offset--;
01285     /* loop through the comma separated array elements */
01286     do {
01287         /* allocate next item */
01288         cJSON *new_item = cJSON_New_Item(&(input_buffer->hooks));
01289         if (new_item == NULL) {
01290             goto fail; /* allocation failure */
01291         }
01292 
01293         /* attach next item to list */
01294         if (head == NULL) {
01295             /* start the linked list */
01296             current_item = head = new_item;
01297         } else {
01298             /* add to the end and advance */
01299             current_item->next = new_item;
01300             new_item->prev = current_item;
01301             current_item = new_item;
01302         }
01303 
01304         /* parse next value */
01305         input_buffer->offset++;
01306         buffer_skip_whitespace(input_buffer);
01307         if (!parse_value(current_item, input_buffer)) {
01308             goto fail; /* failed to parse value */
01309         }
01310         buffer_skip_whitespace(input_buffer);
01311     } while (can_access_at_index(input_buffer, 0) && (buffer_at_offset(input_buffer)[0] == ',));
01312 
01313     if (cannot_access_at_index(input_buffer, 0) || buffer_at_offset(input_buffer)[0] != ']') {
01314         goto fail; /* expected end of array */
01315     }
01316 
01317 success:
01318     input_buffer->depth--;
01319 
01320     if (head != NULL) {
01321         head->prev = current_item;
01322     }
01323 
01324     item->type = cJSON_Array;
01325     item->child = head;
01326 
01327     input_buffer->offset++;
01328 
01329     return true;
01330 
01331 fail:
01332     if (head != NULL) {
01333         cJSON_Delete(head);
01334     }
01335 
01336     return false;
01337 }
01338 
01339 /* Render an array to text */
01340 static cJSON_bool print_array(const cJSON * const item, printbuffer * const output_buffer) {
01341     unsigned char *output_pointer = NULL;
01342     size_t length = 0;
01343     cJSON *current_element = item->child;
01344 
01345     if (output_buffer == NULL) {
01346         return false;
01347     }
01348 
01349     /* Compose the output array.  */
01350     /* opening square bracket */
01351     output_pointer = ensure(output_buffer, 1);
01352     if (output_pointer == NULL) {
01353         return false;
01354     }
01355 
01356     *output_pointer = '[';
01357     output_buffer->offset++;
01358     output_buffer->depth++;
01359 
01360     while (current_element != NULL) {
01361         if (!print_value(current_element, output_buffer)) {
01362             return false;
01363         }
01364         update_offset(output_buffer);
01365         if (current_element->next) {
01366             length = (size_t) (output_buffer->format ?  2 :  1);
01367             output_pointer = ensure(output_buffer, length + 1);
01368             if (output_pointer == NULL) {
01369                 return false;
```

```
01370             }
01371             *output_pointer++ = ',';
01372             if(output_buffer->format) {
01373                 *output_pointer++ = ' ';
01374             }
01375             *output_pointer = '\0';
01376             output_buffer->offset += length;
01377         }
01378         current_element = current_element->next;
01379     }
01380
01381     output_pointer = ensure(output_buffer, 2);
01382     if (output_pointer == NULL) {
01383         return false;
01384     }
01385     *output_pointer++ = ']';
01386     *output_pointer = '\0';
01387     output_buffer->depth--;
01388
01389     return true;
01390 }
01391
01392 /* Build an object from the text.  */
01393 static cJSON_bool parse_object(cJSON * const item, parse_buffer * const input_buffer) {
01394     cJSON *head = NULL; /* linked list head */
01395     cJSON *current_item = NULL;
01396
01397     if (input_buffer->depth >= CJSON_NESTING_LIMIT) {
01398         return false; /* to deeply nested */
01399     }
01400     input_buffer->depth++;
01401
01402     if (cannot_access_at_index(input_buffer, 0) || (buffer_at_offset(input_buffer)[0] != '{')) {
01403         goto fail; /* not an object */
01404     }
01405
01406     input_buffer->offset++;
01407     buffer_skip_whitespace(input_buffer);
01408     if (can_access_at_index(input_buffer, 0) && (buffer_at_offset(input_buffer)[0] == '}')) {
01409         goto success; /* empty object */
01410     }
01411
01412     /* check if we skipped to the end of the buffer */
01413     if (cannot_access_at_index(input_buffer, 0)) {
01414         input_buffer->offset--;
01415         goto fail;
01416     }
01417
01418     /* step back to character in front of the first element */
01419     input_buffer->offset--;
01420     /* loop through the comma separated array elements */
01421     do {
01422         /* allocate next item */
01423         cJSON *new_item = cJSON_New_Item(&(input_buffer->hooks));
01424         if (new_item == NULL) {
01425             goto fail; /* allocation failure */
01426         }
01427
01428         /* attach next item to list */
01429         if (head == NULL) {
01430             /* start the linked list */
01431             current_item = head = new_item;
01432         } else {
01433             /* add to the end and advance */
01434             current_item->next = new_item;
01435             new_item->prev = current_item;
01436             current_item = new_item;
01437         }
01438
01439         if (cannot_access_at_index(input_buffer, 1)) {
01440             goto fail; /* nothing comes after the comma */
01441         }
01442
01443         /* parse the name of the child */
01444         input_buffer->offset++;
01445         buffer_skip_whitespace(input_buffer);
01446         if (!parse_string(current_item, input_buffer)) {
01447             goto fail; /* failed to parse name */
01448         }
01449         buffer_skip_whitespace(input_buffer);
01450
01451         /* swap valuestring and string, because we parsed the name */
01452         current_item->string = current_item->valuestring;
01453         current_item->valuestring = NULL;
01454
01455         if (cannot_access_at_index(input_buffer, 0) || (buffer_at_offset(input_buffer)[0] != ':'))  {
01456             goto fail; /* invalid object */
```

```
01457         }
01458
01459         /* parse the value */
01460         input_buffer->offset++;
01461         buffer_skip_whitespace(input_buffer);
01462         if (!parse_value(current_item, input_buffer)) {
01463             goto fail; /* failed to parse value */
01464         }
01465         buffer_skip_whitespace(input_buffer);
01466     } while (can_access_at_index(input_buffer, 0) && (buffer_at_offset(input_buffer)[0] == ','));
01467
01468     if (cannot_access_at_index(input_buffer, 0) || (buffer_at_offset(input_buffer)[0] != '}')) {
01469         goto fail; /* expected end of object */
01470     }
01471
01472 success:
01473     input_buffer->depth--;
01474
01475     if (head != NULL) {
01476         head->prev = current_item;
01477     }
01478
01479     item->type = cJSON_Object;
01480     item->child = head;
01481
01482     input_buffer->offset++;
01483     return true;
01484
01485 fail:
01486     if (head != NULL) {
01487         cJSON_Delete(head);
01488     }
01489
01490     return false;
01491 }
01492
01493 /* Render an object to text.  */
01494 static cJSON_bool print_object(const cJSON * const item, printbuffer * const output_buffer) {
01495     unsigned char *output_pointer = NULL;
01496     size_t length = 0;
01497     cJSON *current_item = item->child;
01498
01499     if (output_buffer == NULL) {
01500         return false;
01501     }
01502
01503     /* Compose the output:  */
01504     length = (size_t) (output_buffer->format ?  2 :  1); /* fmt:  {\n */
01505     output_pointer = ensure(output_buffer, length + 1);
01506     if (output_pointer == NULL) {
01507         return false;
01508     }
01509
01510     *output_pointer++ = '{';
01511     output_buffer->depth++;
01512     if (output_buffer->format) {
01513         *output_pointer++ = '\n';
01514     }
01515     output_buffer->offset += length;
01516
01517     while (current_item) {
01518         if (output_buffer->format) {
01519             size_t i;
01520             output_pointer = ensure(output_buffer, output_buffer->depth);
01521             if (output_pointer == NULL) {
01522                 return false;
01523             }
01524             for (i = 0; i < output_buffer->depth; i++) {
01525                 *output_pointer++ = '\t';
01526             }
01527             output_buffer->offset += output_buffer->depth;
01528         }
01529
01530         /* print key */
01531         if (!print_string_ptr((unsigned char*)current_item->string, output_buffer)) {
01532             return false;
01533         }
01534         update_offset(output_buffer);
01535
01536         length = (size_t) (output_buffer->format ?  2 :  1);
01537         output_pointer = ensure(output_buffer, length);
01538         if (output_pointer == NULL) {
01539             return false;
01540         }
01541         *output_pointer++ = ':';
01542         if (output_buffer->format) {
01543             *output_pointer++ = '\t';
```

```
01544         }
01545         output_buffer->offset += length;
01546
01547         /* print value */
01548         if (!print_value(current_item, output_buffer)) {
01549             return false;
01550         }
01551         update_offset(output_buffer);
01552
01553         /* print comma if not last */
01554         length = ((size_t)(output_buffer->format ?  1 :  0) + (size_t)(current_item->next ?  1 :  0));
01555         output_pointer = ensure(output_buffer, length + 1);
01556         if (output_pointer == NULL) {
01557             return false;
01558         }
01559         if (current_item->next) {
01560             *output_pointer++ = ',';
01561         }
01562
01563         if (output_buffer->format) {
01564             *output_pointer++ = '\n';
01565         }
01566         *output_pointer = '\0';
01567         output_buffer->offset += length;
01568
01569         current_item = current_item->next;
01570     }
01571
01572     output_pointer = ensure(output_buffer, output_buffer->format ?  (output_buffer->depth + 1) :  2);
01573     if (output_pointer == NULL) {
01574         return false;
01575     }
01576     if (output_buffer->format) {
01577         size_t i;
01578         for (i = 0; i < (output_buffer->depth - 1); i++) {
01579             *output_pointer++ = '\t';
01580         }
01581     }
01582     *output_pointer++ = '}';
01583     *output_pointer = '\0';
01584     output_buffer->depth--;
01585
01586     return true;
01587 }
01588
01589 /* Get Array size/item / object item.  */
01590 CJSON_PUBLIC(int) cJSON_GetArraySize(const cJSON *array) {
01591     cJSON *child = NULL;
01592     size_t size = 0;
01593
01594     if (array == NULL) {
01595         return 0;
01596     }
01597
01598     child = array->child;
01599
01600     while(child != NULL) {
01601         size++;
01602         child = child->next;
01603     }
01604
01605     /* FIXME: Can overflow here.  Cannot be fixed without breaking the API */
01606
01607     return (int)size;
01608 }
01609
01610 static cJSON* get_array_item(const cJSON *array, size_t index) {
01611     cJSON *current_child = NULL;
01612
01613     if (array == NULL) {
01614         return NULL;
01615     }
01616
01617     current_child = array->child;
01618     while ((current_child != NULL) && (index > 0)) {
01619         index--;
01620         current_child = current_child->next;
01621     }
01622
01623     return current_child;
01624 }
01625
01626 CJSON_PUBLIC(cJSON *) cJSON_GetArrayItem(const cJSON *array, int index) {
01627     if (index < 0) {
01628         return NULL;
01629     }
01630
```

```
01631     return get_array_item(array, (size_t)index);
01632 }
01633
01634 static cJSON *get_object_item(const cJSON * const object, const char * const name, const cJSON_bool
      case_sensitive) {
01635     cJSON *current_element = NULL;
01636
01637     if ((object == NULL) || (name == NULL)) {
01638         return NULL;
01639     }
01640
01641     current_element = object->child;
01642     if (case_sensitive) {
01643         while ((current_element != NULL) && (current_element->string != NULL) && (strcmp(name,
      current_element->string) != 0)) {
01644             current_element = current_element->next;
01645         }
01646     } else {
01647         while ((current_element != NULL) && (case_insensitive_strcmp((const unsigned char*)name,
      (const unsigned char*)(current_element->string)) != 0)) {
01648             current_element = current_element->next;
01649         }
01650     }
01651
01652     if ((current_element == NULL) || (current_element->string == NULL)) {
01653         return NULL;
01654     }
01655
01656     return current_element;
01657 }
01658
01659 CJSON_PUBLIC(cJSON *) cJSON_GetObjectItem(const cJSON * const object, const char * const string) {
01660     return get_object_item(object, string, false);
01661 }
01662
01663 CJSON_PUBLIC(cJSON *) cJSON_GetObjectItemCaseSensitive(const cJSON * const object, const char * const
      string) {
01664     return get_object_item(object, string, true);
01665 }
01666
01667 CJSON_PUBLIC(cJSON_bool) cJSON_HasObjectItem(const cJSON *object, const char *string) {
01668     return cJSON_GetObjectItem(object, string) ?  1 :  0;
01669 }
01670
01671 /* Utility for array list handling.  */
01672 static void suffix_object(cJSON *prev, cJSON *item) {
01673     prev->next = item;
01674     item->prev = prev;
01675 }
01676
01677 /* Utility for handling references.  */
01678 static cJSON *create_reference(const cJSON *item, const internal_hooks * const hooks) {
01679     cJSON *reference = NULL;
01680     if (item == NULL) {
01681         return NULL;
01682     }
01683
01684     reference = cJSON_New_Item(hooks);
01685     if (reference == NULL) {
01686         return NULL;
01687     }
01688
01689     memcpy(reference, item, sizeof(cJSON));
01690     reference->string = NULL;
01691     reference->type |= cJSON_IsReference;
01692     reference->next = reference->prev = NULL;
01693     return reference;
01694 }
01695
01696 static cJSON_bool add_item_to_array(cJSON *array, cJSON *item) {
01697     cJSON *child = NULL;
01698
01699     if ((item == NULL) || (array == NULL) || (array == item)) {
01700         return false;
01701     }
01702
01703     child = array->child;
01704     /*
01705 * To find the last item in array quickly, we use prev in array
01706 */
01707     if (child == NULL) {
01708         /* list is empty, start new one */
01709         array->child = item;
01710         item->prev = item;
01711         item->next = NULL;
01712     } else {
01713         /* append to the end */
```

```
01714         if (child->prev) {
01715             suffix_object(child->prev, item);
01716             array->child->prev = item;
01717         }
01718     }
01719
01720     return true;
01721 }
01722
01723 /* Add item to array/object.  */
01724 CJSON_PUBLIC(cJSON_bool) cJSON_AddItemToArray(cJSON *array, cJSON *item) {
01725     return add_item_to_array(array, item);
01726 }
01727
01728 #if defined(__clang__) || (defined(__GNUC__)  && ((__GNUC__ > 4) || ((__GNUC__ == 4) &&
      (__GNUC_MINOR__ > 5))))
01729 #pragma GCC diagnostic push
01730 #endif
01731 #ifdef __GNUC__
01732 #pragma GCC diagnostic ignored "-Wcast-qual"
01733 #endif
01734 /* helper function to cast away const */
01735 static void* cast_away_const(const void* string) {
01736     return (void*)string;
01737 }
01738 #if defined(__clang__) || (defined(__GNUC__)  && ((__GNUC__ > 4) || ((__GNUC__ == 4) &&
      (__GNUC_MINOR__ > 5))))
01739 #pragma GCC diagnostic pop
01740 #endif
01741
01742
01743 static cJSON_bool add_item_to_object(cJSON * const object, const char * const string, cJSON * const
      item, const internal_hooks * const hooks, const cJSON_bool constant_key) {
01744     char *new_key = NULL;
01745     int new_type = cJSON_Invalid;
01746
01747     if ((object == NULL) || (string == NULL) || (item == NULL) || (object == item)) {
01748         return false;
01749     }
01750
01751     if (constant_key) {
01752         new_key = (char*)cast_away_const(string);
01753         new_type = item->type | cJSON_StringIsConst;
01754     } else {
01755         new_key = (char*)cJSON_strdup((const unsigned char*)string, hooks);
01756         if (new_key == NULL) {
01757             return false;
01758         }
01759
01760         new_type = item->type & ~cJSON_StringIsConst;
01761     }
01762
01763     if (!(item->type & cJSON_StringIsConst) && (item->string != NULL)) {
01764         hooks->deallocate(item->string);
01765     }
01766
01767     item->string = new_key;
01768     item->type = new_type;
01769
01770     return add_item_to_array(object, item);
01771 }
01772
01773 CJSON_PUBLIC(cJSON_bool) cJSON_AddItemToObject(cJSON *object, const char *string, cJSON *item) {
01774     return add_item_to_object(object, string, item, &global_hooks, false);
01775 }
01776
01777 /* Add an item to an object with constant string as key */
01778 CJSON_PUBLIC(cJSON_bool) cJSON_AddItemToObjectCS(cJSON *object, const char *string, cJSON *item) {
01779     return add_item_to_object(object, string, item, &global_hooks, true);
01780 }
01781
01782 CJSON_PUBLIC(cJSON_bool) cJSON_AddItemReferenceToArray(cJSON *array, cJSON *item) {
01783     if (array == NULL) {
01784         return false;
01785     }
01786
01787     return add_item_to_array(array, create_reference(item, &global_hooks));
01788 }
01789
01790 CJSON_PUBLIC(cJSON_bool) cJSON_AddItemReferenceToObject(cJSON *object, const char *string, cJSON
      *item) {
01791     if ((object == NULL) || (string == NULL)) {
01792         return false;
01793     }
01794
01795     return add_item_to_object(object, string, create_reference(item, &global_hooks), &global_hooks,
      false);
```

```
01796 }
01797
01798 CJSON_PUBLIC(cJSON*) cJSON_AddNullToObject(cJSON * const object, const char * const name) {
01799     cJSON *null = cJSON_CreateNull();
01800     if (add_item_to_object(object, name, null, &global_hooks, false)) {
01801         return null;
01802     }
01803
01804     cJSON_Delete(null);
01805     return NULL;
01806 }
01807
01808 CJSON_PUBLIC(cJSON*) cJSON_AddTrueToObject(cJSON * const object, const char * const name) {
01809     cJSON *true_item = cJSON_CreateTrue();
01810     if (add_item_to_object(object, name, true_item, &global_hooks, false)) {
01811         return true_item;
01812     }
01813
01814     cJSON_Delete(true_item);
01815     return NULL;
01816 }
01817
01818 CJSON_PUBLIC(cJSON*) cJSON_AddFalseToObject(cJSON * const object, const char * const name) {
01819     cJSON *false_item = cJSON_CreateFalse();
01820     if (add_item_to_object(object, name, false_item, &global_hooks, false)) {
01821         return false_item;
01822     }
01823
01824     cJSON_Delete(false_item);
01825     return NULL;
01826 }
01827
01828 CJSON_PUBLIC(cJSON*) cJSON_AddBoolToObject(cJSON * const object, const char * const name, const
      cJSON_bool boolean) {
01829     cJSON *bool_item = cJSON_CreateBool(boolean);
01830     if (add_item_to_object(object, name, bool_item, &global_hooks, false)) {
01831         return bool_item;
01832     }
01833
01834     cJSON_Delete(bool_item);
01835     return NULL;
01836 }
01837
01838 CJSON_PUBLIC(cJSON*) cJSON_AddNumberToObject(cJSON * const object, const char * const name, const
      double number) {
01839     cJSON *number_item = cJSON_CreateNumber(number);
01840     if (add_item_to_object(object, name, number_item, &global_hooks, false)) {
01841         return number_item;
01842     }
01843
01844     cJSON_Delete(number_item);
01845     return NULL;
01846 }
01847
01848 CJSON_PUBLIC(cJSON*) cJSON_AddStringToObject(cJSON * const object, const char * const name, const char
      * const string) {
01849     cJSON *string_item = cJSON_CreateString(string);
01850     if (add_item_to_object(object, name, string_item, &global_hooks, false)) {
01851         return string_item;
01852     }
01853
01854     cJSON_Delete(string_item);
01855     return NULL;
01856 }
01857
01858 CJSON_PUBLIC(cJSON*) cJSON_AddRawToObject(cJSON * const object, const char * const name, const char *
      const raw) {
01859     cJSON *raw_item = cJSON_CreateRaw(raw);
01860     if (add_item_to_object(object, name, raw_item, &global_hooks, false)) {
01861         return raw_item;
01862     }
01863
01864     cJSON_Delete(raw_item);
01865     return NULL;
01866 }
01867
01868 CJSON_PUBLIC(cJSON*) cJSON_AddObjectToObject(cJSON * const object, const char * const name) {
01869     cJSON *object_item = cJSON_CreateObject();
01870     if (add_item_to_object(object, name, object_item, &global_hooks, false)) {
01871         return object_item;
01872     }
01873
01874     cJSON_Delete(object_item);
01875     return NULL;
01876 }
01877
01878 CJSON_PUBLIC(cJSON*) cJSON_AddArrayToObject(cJSON * const object, const char * const name) {
```

```
01879     cJSON *array = cJSON_CreateArray();
01880     if (add_item_to_object(object, name, array, &global_hooks, false)) {
01881         return array;
01882     }
01883 
01884     cJSON_Delete(array);
01885     return NULL;
01886 }
01887 
01888 CJSON_PUBLIC(cJSON *) cJSON_DetachItemViaPointer(cJSON *parent, cJSON * const item) {
01889     if ((parent == NULL) || (item == NULL) || (item != parent->child && item->prev == NULL)) {
01890         return NULL;
01891     }
01892 
01893     if (item != parent->child) {
01894         /* not the first element */
01895         item->prev->next = item->next;
01896     }
01897     if (item->next != NULL) {
01898         /* not the last element */
01899         item->next->prev = item->prev;
01900     }
01901 
01902     if (item == parent->child) {
01903         /* first element */
01904         parent->child = item->next;
01905     } else if (item->next == NULL) {
01906         /* last element */
01907         parent->child->prev = item->prev;
01908     }
01909 
01910     /* make sure the detached item doesn't point anywhere anymore */
01911     item->prev = NULL;
01912     item->next = NULL;
01913 
01914     return item;
01915 }
01916 
01917 CJSON_PUBLIC(cJSON *) cJSON_DetachItemFromArray(cJSON *array, int which) {
01918     if (which < 0) {
01919         return NULL;
01920     }
01921 
01922     return cJSON_DetachItemViaPointer(array, get_array_item(array, (size_t)which));
01923 }
01924 
01925 CJSON_PUBLIC(void) cJSON_DeleteItemFromArray(cJSON *array, int which) {
01926     cJSON_Delete(cJSON_DetachItemFromArray(array, which));
01927 }
01928 
01929 CJSON_PUBLIC(cJSON *) cJSON_DetachItemFromObject(cJSON *object, const char *string) {
01930     cJSON *to_detach = cJSON_GetObjectItem(object, string);
01931 
01932     return cJSON_DetachItemViaPointer(object, to_detach);
01933 }
01934 
01935 CJSON_PUBLIC(cJSON *) cJSON_DetachItemFromObjectCaseSensitive(cJSON *object, const char *string) {
01936     cJSON *to_detach = cJSON_GetObjectItemCaseSensitive(object, string);
01937 
01938     return cJSON_DetachItemViaPointer(object, to_detach);
01939 }
01940 
01941 CJSON_PUBLIC(void) cJSON_DeleteItemFromObject(cJSON *object, const char *string) {
01942     cJSON_Delete(cJSON_DetachItemFromObject(object, string));
01943 }
01944 
01945 CJSON_PUBLIC(void) cJSON_DeleteItemFromObjectCaseSensitive(cJSON *object, const char *string) {
01946     cJSON_Delete(cJSON_DetachItemFromObjectCaseSensitive(object, string));
01947 }
01948 
01949 /* Replace array/object items with new ones.  */
01950 CJSON_PUBLIC(cJSON_bool) cJSON_InsertItemInArray(cJSON *array, int which, cJSON *newitem) {
01951     cJSON *after_inserted = NULL;
01952 
01953     if (which < 0 || newitem == NULL) {
01954         return false;
01955     }
01956 
01957     after_inserted = get_array_item(array, (size_t)which);
01958     if (after_inserted == NULL) {
01959         return add_item_to_array(array, newitem);
01960     }
01961 
01962     if (after_inserted != array->child && after_inserted->prev == NULL) {
01963         /* return false if after_inserted is a corrupted array item */
01964         return false;
01965     }
```

```
01966
01967     newitem->next = after_inserted;
01968     newitem->prev = after_inserted->prev;
01969     after_inserted->prev = newitem;
01970     if (after_inserted == array->child) {
01971         array->child = newitem;
01972     } else {
01973         newitem->prev->next = newitem;
01974     }
01975     return true;
01976 }
01977
01978 CJSON_PUBLIC(cJSON_bool) cJSON_ReplaceItemViaPointer(cJSON * const parent, cJSON * const item, cJSON *
      replacement) {
01979     if ((parent == NULL) || (parent->child == NULL) || (replacement == NULL) || (item == NULL)) {
01980         return false;
01981     }
01982
01983     if (replacement == item) {
01984         return true;
01985     }
01986
01987     replacement->next = item->next;
01988     replacement->prev = item->prev;
01989
01990     if (replacement->next != NULL) {
01991         replacement->next->prev = replacement;
01992     }
01993     if (parent->child == item) {
01994         if (parent->child->prev == parent->child) {
01995             replacement->prev = replacement;
01996         }
01997         parent->child = replacement;
01998     } else {
01999         /*
02000 * To find the last item in array quickly, we use prev in array.
02001 * We can't modify the last item's next pointer where this item was the parent's child
02002 */
02003         if (replacement->prev != NULL) {
02004             replacement->prev->next = replacement;
02005         }
02006         if (replacement->next == NULL) {
02007             parent->child->prev = replacement;
02008         }
02009     }
02010
02011     item->next = NULL;
02012     item->prev = NULL;
02013     cJSON_Delete(item);
02014
02015     return true;
02016 }
02017
02018 CJSON_PUBLIC(cJSON_bool) cJSON_ReplaceItemInArray(cJSON *array, int which, cJSON *newitem) {
02019     if (which < 0) {
02020         return false;
02021     }
02022
02023     return cJSON_ReplaceItemViaPointer(array, get_array_item(array, (size_t)which), newitem);
02024 }
02025
02026 static cJSON_bool replace_item_in_object(cJSON *object, const char *string, cJSON *replacement,
      cJSON_bool case_sensitive) {
02027     if ((replacement == NULL) || (string == NULL)) {
02028         return false;
02029     }
02030
02031     /* replace the name in the replacement */
02032     if (!(replacement->type & cJSON_StringIsConst) && (replacement->string != NULL)) {
02033         cJSON_free(replacement->string);
02034     }
02035     replacement->string = (char*)cJSON_strdup((const unsigned char*)string, &global_hooks);
02036     if (replacement->string == NULL) {
02037         return false;
02038     }
02039
02040     replacement->type &= ~cJSON_StringIsConst;
02041
02042     return cJSON_ReplaceItemViaPointer(object, get_object_item(object, string, case_sensitive),
      replacement);
02043 }
02044
02045 CJSON_PUBLIC(cJSON_bool) cJSON_ReplaceItemInObject(cJSON *object, const char *string, cJSON *newitem)
      {
02046     return replace_item_in_object(object, string, newitem, false);
02047 }
02048
```

```
02049 CJSON_PUBLIC(cJSON_bool) cJSON_ReplaceItemInObjectCaseSensitive(cJSON *object, const char *string,
      cJSON *newitem) {
02050     return replace_item_in_object(object, string, newitem, true);
02051 }
02052
02053 /* Create basic types:  */
02054 CJSON_PUBLIC(cJSON *) cJSON_CreateNull(void) {
02055     cJSON *item = cJSON_New_Item(&global_hooks);
02056     if(item) {
02057         item->type = cJSON_NULL;
02058     }
02059
02060     return item;
02061 }
02062
02063 CJSON_PUBLIC(cJSON *) cJSON_CreateTrue(void) {
02064     cJSON *item = cJSON_New_Item(&global_hooks);
02065     if(item) {
02066         item->type = cJSON_True;
02067     }
02068
02069     return item;
02070 }
02071
02072 CJSON_PUBLIC(cJSON *) cJSON_CreateFalse(void) {
02073     cJSON *item = cJSON_New_Item(&global_hooks);
02074     if(item) {
02075         item->type = cJSON_False;
02076     }
02077
02078     return item;
02079 }
02080
02081 CJSON_PUBLIC(cJSON *) cJSON_CreateBool(cJSON_bool boolean) {
02082     cJSON *item = cJSON_New_Item(&global_hooks);
02083     if(item) {
02084         item->type = boolean ?  cJSON_True :  cJSON_False;
02085     }
02086
02087     return item;
02088 }
02089
02090 CJSON_PUBLIC(cJSON *) cJSON_CreateNumber(double num) {
02091     cJSON *item = cJSON_New_Item(&global_hooks);
02092     if(item) {
02093         item->type = cJSON_Number;
02094         item->valuedouble = num;
02095
02096         /* use saturation in case of overflow */
02097         if (num >= INT_MAX) {
02098             item->valueint = INT_MAX;
02099         } else if (num <= (double)INT_MIN) {
02100             item->valueint = INT_MIN;
02101         } else {
02102             item->valueint = (int)num;
02103         }
02104     }
02105
02106     return item;
02107 }
02108
02109 CJSON_PUBLIC(cJSON *) cJSON_CreateString(const char *string) {
02110     cJSON *item = cJSON_New_Item(&global_hooks);
02111     if(item) {
02112         item->type = cJSON_String;
02113         item->valuestring = (char*)cJSON_strdup((const unsigned char*)string, &global_hooks);
02114         if(!item->valuestring) {
02115             cJSON_Delete(item);
02116             return NULL;
02117         }
02118     }
02119
02120     return item;
02121 }
02122
02123 CJSON_PUBLIC(cJSON *) cJSON_CreateStringReference(const char *string) {
02124     cJSON *item = cJSON_New_Item(&global_hooks);
02125     if (item != NULL) {
02126         item->type = cJSON_String | cJSON_IsReference;
02127         item->valuestring = (char*)cast_away_const(string);
02128     }
02129
02130     return item;
02131 }
02132
02133 CJSON_PUBLIC(cJSON *) cJSON_CreateObjectReference(const cJSON *child) {
02134     cJSON *item = cJSON_New_Item(&global_hooks);
```

```
02135     if (item != NULL) {
02136         item->type = cJSON_Object | cJSON_IsReference;
02137         item->child = (cJSON*)cast_away_const(child);
02138     }
02139 
02140     return item;
02141 }
02142 
02143 CJSON_PUBLIC(cJSON *) cJSON_CreateArrayReference(const cJSON *child) {
02144     cJSON *item = cJSON_New_Item(&global_hooks);
02145     if (item != NULL) {
02146         item->type = cJSON_Array | cJSON_IsReference;
02147         item->child = (cJSON*)cast_away_const(child);
02148     }
02149 
02150     return item;
02151 }
02152 
02153 CJSON_PUBLIC(cJSON *) cJSON_CreateRaw(const char *raw) {
02154     cJSON *item = cJSON_New_Item(&global_hooks);
02155     if(item) {
02156         item->type = cJSON_Raw;
02157         item->valuestring = (char*)cJSON_strdup((const unsigned char*)raw, &global_hooks);
02158         if(!item->valuestring) {
02159             cJSON_Delete(item);
02160             return NULL;
02161         }
02162     }
02163 
02164     return item;
02165 }
02166 
02167 CJSON_PUBLIC(cJSON *) cJSON_CreateArray(void) {
02168     cJSON *item = cJSON_New_Item(&global_hooks);
02169     if(item) {
02170         item->type=cJSON_Array;
02171     }
02172 
02173     return item;
02174 }
02175 
02176 CJSON_PUBLIC(cJSON *) cJSON_CreateObject(void) {
02177     cJSON *item = cJSON_New_Item(&global_hooks);
02178     if (item) {
02179         item->type = cJSON_Object;
02180     }
02181 
02182     return item;
02183 }
02184 
02185 /* Create Arrays:  */
02186 CJSON_PUBLIC(cJSON *) cJSON_CreateIntArray(const int *numbers, int count) {
02187     size_t i = 0;
02188     cJSON *n = NULL;
02189     cJSON *p = NULL;
02190     cJSON *a = NULL;
02191 
02192     if ((count < 0) || (numbers == NULL)) {
02193         return NULL;
02194     }
02195 
02196     a = cJSON_CreateArray();
02197 
02198     for(i = 0; a && (i < (size_t)count); i++) {
02199         n = cJSON_CreateNumber(numbers[i]);
02200         if (!n) {
02201             cJSON_Delete(a);
02202             return NULL;
02203         }
02204         if(!i) {
02205             a->child = n;
02206         } else {
02207             suffix_object(p, n);
02208         }
02209         p = n;
02210     }
02211 
02212     if (a && a->child) {
02213         a->child->prev = n;
02214     }
02215 
02216     return a;
02217 }
02218 
02219 CJSON_PUBLIC(cJSON *) cJSON_CreateFloatArray(const float *numbers, int count) {
02220     size_t i = 0;
02221     cJSON *n = NULL;
```

```
02222     cJSON *p = NULL;
02223     cJSON *a = NULL;
02224
02225     if ((count < 0) || (numbers == NULL)) {
02226         return NULL;
02227     }
02228
02229     a = cJSON_CreateArray();
02230
02231     for(i = 0; a && (i < (size_t)count); i++) {
02232         n = cJSON_CreateNumber((double)numbers[i]);
02233         if(!n) {
02234             cJSON_Delete(a);
02235             return NULL;
02236         }
02237         if(!i) {
02238             a->child = n;
02239         } else {
02240             suffix_object(p, n);
02241         }
02242         p = n;
02243     }
02244
02245     if (a && a->child) {
02246         a->child->prev = n;
02247     }
02248
02249     return a;
02250 }
02251
02252 CJSON_PUBLIC(cJSON *) cJSON_CreateDoubleArray(const double *numbers, int count) {
02253     size_t i = 0;
02254     cJSON *n = NULL;
02255     cJSON *p = NULL;
02256     cJSON *a = NULL;
02257
02258     if ((count < 0) || (numbers == NULL)) {
02259         return NULL;
02260     }
02261
02262     a = cJSON_CreateArray();
02263
02264     for(i = 0; a && (i < (size_t)count); i++) {
02265         n = cJSON_CreateNumber(numbers[i]);
02266         if(!n) {
02267             cJSON_Delete(a);
02268             return NULL;
02269         }
02270         if(!i) {
02271             a->child = n;
02272         } else {
02273             suffix_object(p, n);
02274         }
02275         p = n;
02276     }
02277
02278     if (a && a->child) {
02279         a->child->prev = n;
02280     }
02281
02282     return a;
02283 }
02284
02285 CJSON_PUBLIC(cJSON *) cJSON_CreateStringArray(const char *const *strings, int count) {
02286     size_t i = 0;
02287     cJSON *n = NULL;
02288     cJSON *p = NULL;
02289     cJSON *a = NULL;
02290
02291     if ((count < 0) || (strings == NULL)) {
02292         return NULL;
02293     }
02294
02295     a = cJSON_CreateArray();
02296
02297     for (i = 0; a && (i < (size_t)count); i++) {
02298         n = cJSON_CreateString(strings[i]);
02299         if(!n) {
02300             cJSON_Delete(a);
02301             return NULL;
02302         }
02303         if(!i) {
02304             a->child = n;
02305         } else {
02306             suffix_object(p,n);
02307         }
02308         p = n;
```

```
02309     }
02310
02311     if (a && a->child) {
02312         a->child->prev = n;
02313     }
02314
02315     return a;
02316 }
02317
02318 /* Duplication */
02319 cJSON * cJSON_Duplicate_rec(const cJSON *item, size_t depth, cJSON_bool recurse);
02320
02321 CJSON_PUBLIC(cJSON *) cJSON_Duplicate(const cJSON *item, cJSON_bool recurse) {
02322     return cJSON_Duplicate_rec(item, 0, recurse );
02323 }
02324
02325 cJSON * cJSON_Duplicate_rec(const cJSON *item, size_t depth, cJSON_bool recurse) {
02326     cJSON *newitem = NULL;
02327     cJSON *child = NULL;
02328     cJSON *next = NULL;
02329     cJSON *newchild = NULL;
02330
02331     /* Bail on bad ptr */
02332     if (!item) {
02333         goto fail;
02334     }
02335     /* Create new item */
02336     newitem = cJSON_New_Item(&global_hooks);
02337     if (!newitem) {
02338         goto fail;
02339     }
02340     /* Copy over all vars */
02341     newitem->type = item->type & (~cJSON_IsReference);
02342     newitem->valueint = item->valueint;
02343     newitem->valuedouble = item->valuedouble;
02344     if (item->valuestring) {
02345         newitem->valuestring = (char*)cJSON_strdup((unsigned char*)item->valuestring, &global_hooks);
02346         if (!newitem->valuestring) {
02347             goto fail;
02348         }
02349     }
02350     if (item->string) {
02351         newitem->string = (item->type&cJSON_StringIsConst) ?  item->string :
      (char*)cJSON_strdup((unsigned char*)item->string, &global_hooks);
02352         if (!newitem->string) {
02353             goto fail;
02354         }
02355     }
02356     /* If non-recursive, then we're done!  */
02357     if (!recurse) {
02358         return newitem;
02359     }
02360     /* Walk the ->next chain for the child.  */
02361     child = item->child;
02362     while (child != NULL) {
02363         if(depth >= CJSON_CIRCULAR_LIMIT) {
02364             goto fail;
02365         }
02366         newchild = cJSON_Duplicate_rec(child, depth + 1, true); /* Duplicate (with recurse) each item
      in the ->next chain */
02367         if (!newchild) {
02368             goto fail;
02369         }
02370         if (next != NULL) {
02371             /* If newitem->child already set, then crosswire ->prev and ->next and move on */
02372             next->next = newchild;
02373             newchild->prev = next;
02374             next = newchild;
02375         } else {
02376             /* Set newitem->child and move to it */
02377             newitem->child = newchild;
02378             next = newchild;
02379         }
02380         child = child->next;
02381     }
02382     if (newitem && newitem->child) {
02383         newitem->child->prev = newchild;
02384     }
02385
02386     return newitem;
02387
02388 fail:
02389     if (newitem != NULL) {
02390         cJSON_Delete(newitem);
02391     }
02392
02393     return NULL;
```

```
02394 }
02395
02396 static void skip_oneline_comment(char **input) {
02397     *input += static_strlen("//");
02398
02399     for (; (*input)[0] != '\0'; ++(*input)) {
02400         if ((*input)[0] == '\n') {
02401             *input += static_strlen("\n");
02402             return;
02403         }
02404     }
02405 }
02406
02407 static void skip_multiline_comment(char **input) {
02408     *input += static_strlen("/*");
02409
02410     for (; (*input)[0] != '\0'; ++(*input)) {
02411         if (((*input)[0] == '*') && ((*input)[1] == '/')) {
02412             *input += static_strlen("*/");
02413             return;
02414         }
02415     }
02416 }
02417
02418 static void minify_string(char **input, char **output) {
02419     (*output)[0] = (*input)[0];
02420     *input += static_strlen("\"");
02421     *output += static_strlen("\"");
02422
02423
02424     for (; (*input)[0] != '\0'; (void)++(*input), ++(*output)) {
02425         (*output)[0] = (*input)[0];
02426
02427         if ((*input)[0] == '\"') {
02428             (*output)[0] = '\"';
02429             *input += static_strlen("\"");
02430             *output += static_strlen("\"");
02431             return;
02432         } else if (((*input)[0] == '\\') && ((*input)[1] == '\"')) {
02433             (*output)[1] = (*input)[1];
02434             *input += static_strlen("\"");
02435             *output += static_strlen("\"");
02436         }
02437     }
02438 }
02439
02440 CJSON_PUBLIC(void) cJSON_Minify(char *json) {
02441     char *into = json;
02442
02443     if (json == NULL) {
02444         return;
02445     }
02446
02447     while (json[0] != '\0') {
02448         switch (json[0]) {
02449         case ' ':
02450         case '\t':
02451         case '\r':
02452         case '\n':
02453             json++;
02454             break;
02455
02456         case '/':
02457             if (json[1] == '/') {
02458                 skip_oneline_comment(&json);
02459             } else if (json[1] == '*') {
02460                 skip_multiline_comment(&json);
02461             } else {
02462                 json++;
02463             }
02464             break;
02465
02466         case '\"':
02467             minify_string(&json, (char**)&into);
02468             break;
02469
02470         default:
02471             into[0] = json[0];
02472             json++;
02473             into++;
02474         }
02475     }
02476
02477     /* and null-terminate.  */
02478     *into = '\0';
02479 }
02480
```

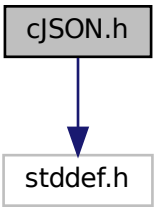
```
02481 CJSON_PUBLIC(cJSON_bool) cJSON_IsInvalid(const cJSON * const item) {
02482     if (item == NULL) {
02483         return false;
02484     }
02485 
02486     return (item->type & 0xFF) == cJSON_Invalid;
02487 }
02488 
02489 CJSON_PUBLIC(cJSON_bool) cJSON_IsFalse(const cJSON * const item) {
02490     if (item == NULL) {
02491         return false;
02492     }
02493 
02494     return (item->type & 0xFF) == cJSON_False;
02495 }
02496 
02497 CJSON_PUBLIC(cJSON_bool) cJSON_IsTrue(const cJSON * const item) {
02498     if (item == NULL) {
02499         return false;
02500     }
02501 
02502     return (item->type & 0xff) == cJSON_True;
02503 }
02504 
02505 
02506 CJSON_PUBLIC(cJSON_bool) cJSON_IsBool(const cJSON * const item) {
02507     if (item == NULL) {
02508         return false;
02509     }
02510 
02511     return (item->type & (cJSON_True | cJSON_False)) != 0;
02512 }
02513 CJSON_PUBLIC(cJSON_bool) cJSON_IsNull(const cJSON * const item) {
02514     if (item == NULL) {
02515         return false;
02516     }
02517 
02518     return (item->type & 0xFF) == cJSON_NULL;
02519 }
02520 
02521 CJSON_PUBLIC(cJSON_bool) cJSON_IsNumber(const cJSON * const item) {
02522     if (item == NULL) {
02523         return false;
02524     }
02525 
02526     return (item->type & 0xFF) == cJSON_Number;
02527 }
02528 
02529 CJSON_PUBLIC(cJSON_bool) cJSON_IsString(const cJSON * const item) {
02530     if (item == NULL) {
02531         return false;
02532     }
02533 
02534     return (item->type & 0xFF) == cJSON_String;
02535 }
02536 
02537 CJSON_PUBLIC(cJSON_bool) cJSON_IsArray(const cJSON * const item) {
02538     if (item == NULL) {
02539         return false;
02540     }
02541 
02542     return (item->type & 0xFF) == cJSON_Array;
02543 }
02544 
02545 CJSON_PUBLIC(cJSON_bool) cJSON_IsObject(const cJSON * const item) {
02546     if (item == NULL) {
02547         return false;
02548     }
02549 
02550     return (item->type & 0xFF) == cJSON_Object;
02551 }
02552 
02553 CJSON_PUBLIC(cJSON_bool) cJSON_IsRaw(const cJSON * const item) {
02554     if (item == NULL) {
02555         return false;
02556     }
02557 
02558     return (item->type & 0xFF) == cJSON_Raw;
02559 }
02560 
02561 CJSON_PUBLIC(cJSON_bool) cJSON_Compare(const cJSON * const a, const cJSON * const b, const cJSON_bool
    case_sensitive) {
02562     if ((a == NULL) || (b == NULL) || ((a->type & 0xFF) != (b->type & 0xFF))) {
02563         return false;
02564     }
02565 
02566     /* check if type is valid */
```

```
02567     switch (a->type & 0xFF) {
02568     case cJSON_False:
02569     case cJSON_True:
02570     case cJSON_NULL:
02571     case cJSON_Number:
02572     case cJSON_String:
02573     case cJSON_Raw:
02574     case cJSON_Array:
02575     case cJSON_Object:
02576         break;
02577
02578     default:
02579         return false;
02580     }
02581
02582     /* identical objects are equal */
02583     if (a == b) {
02584         return true;
02585     }
02586
02587     switch (a->type & 0xFF) {
02588     /* in these cases and equal type is enough */
02589     case cJSON_False:
02590     case cJSON_True:
02591     case cJSON_NULL:
02592         return true;
02593
02594     case cJSON_Number:
02595         if (compare_double(a->valuedouble, b->valuedouble)) {
02596             return true;
02597         }
02598         return false;
02599
02600     case cJSON_String:
02601     case cJSON_Raw:
02602         if ((a->valuestring == NULL) || (b->valuestring == NULL)) {
02603             return false;
02604         }
02605         if (strcmp(a->valuestring, b->valuestring) == 0) {
02606             return true;
02607         }
02608
02609         return false;
02610
02611     case cJSON_Array:  {
02612         cJSON *a_element = a->child;
02613         cJSON *b_element = b->child;
02614
02615         for (; (a_element != NULL) && (b_element != NULL);) {
02616             if (!cJSON_Compare(a_element, b_element, case_sensitive)) {
02617                 return false;
02618             }
02619
02620             a_element = a_element->next;
02621             b_element = b_element->next;
02622         }
02623
02624         /* one of the arrays is longer than the other */
02625         if (a_element != b_element) {
02626             return false;
02627         }
02628
02629         return true;
02630     }
02631
02632     case cJSON_Object:  {
02633         cJSON *a_element = NULL;
02634         cJSON *b_element = NULL;
02635         cJSON_ArrayForEach(a_element, a) {
02636             /* TODO This has O(n^2) runtime, which is horrible!  */
02637             b_element = get_object_item(b, a_element->string, case_sensitive);
02638             if (b_element == NULL) {
02639                 return false;
02640             }
02641
02642             if (!cJSON_Compare(a_element, b_element, case_sensitive)) {
02643                 return false;
02644             }
02645         }
02646
02647         /* doing this twice, once on a and b to prevent true comparison if a subset of b
02648  * TODO: Do this the proper way, this is just a fix for now */
02649         cJSON_ArrayForEach(b_element, b) {
02650             a_element = get_object_item(a, b_element->string, case_sensitive);
02651             if (a_element == NULL) {
02652                 return false;
02653             }
```

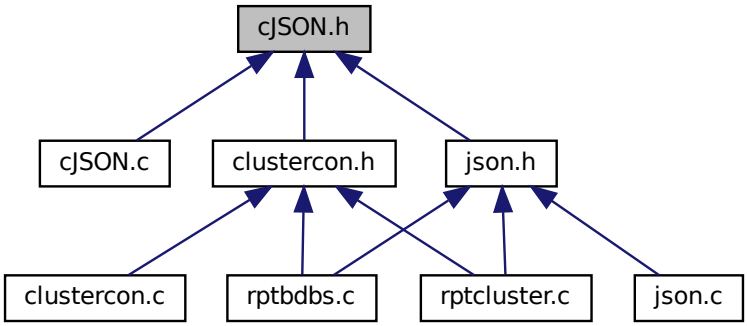
```
02654
02655             if (!cJSON_Compare(b_element, a_element, case_sensitive)) {
02656                 return false;
02657             }
02658         }
02659
02660         return true;
02661     }
02662
02663     default:
02664         return false;
02665     }
02666 }
02667
02668 CJSON_PUBLIC(void *) cJSON_malloc(size_t size) {
02669     return global_hooks.allocate(size);
02670 }
02671
02672 CJSON_PUBLIC(void) cJSON_free(void *object) {
02673     global_hooks.deallocate(object);
02674     object = NULL;
02675 }
```

4.9 cJSON.h File Reference

`#include <stddef.h>`
Include dependency graph for cJSON.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct cJSON
- struct cJSON_Hooks

Macros

- #define CJSON_CDECL
- #define CJSON_STDCALL
- #define CJSON_PUBLIC(type) type
- #define CJSON_VERSION_MAJOR 1
- #define CJSON_VERSION_MINOR 7
- #define CJSON_VERSION_PATCH 18
- #define cJSON_Invalid (0)
- #define cJSON_False (1 << 0)
- #define cJSON_True (1 << 1)
- #define cJSON_NULL (1 << 2)
- #define cJSON_Number (1 << 3)
- #define cJSON_String (1 << 4)
- #define cJSON_Array (1 << 5)
- #define cJSON_Object (1 << 6)
- #define cJSON_Raw (1 << 7) /* raw json */
- #define cJSON_IsReference 256
- #define cJSON_StringIsConst 512
- #define CJSON_NESTING_LIMIT 1000
- #define CJSON_CIRCULAR_LIMIT 10000
- #define cJSON_SetIntValue(object, number) ((object) ? (object)->valueint = (object)->valuedouble = (number) : (number))
- #define cJSON_SetNumberValue(object, number) ((object != NULL) ? cJSON_SetNumberHelper(object, (double)number) : (number))
- #define cJSON_SetBoolValue(object, boolValue)
- #define cJSON_ArrayForEach(element, array) for(element = (array != NULL) ? (array)->child : NULL; element != NULL; element = element->next)

Typedefs

- typedef struct cJSON cJSON
- typedef struct cJSON_Hooks cJSON_Hooks
- typedef int cJSON_bool

Functions

- CJSON_PUBLIC (const char *) cJSON_Version(void)
- CJSON_PUBLIC (void) cJSON_InitHooks(cJSON_Hooks *hooks)
- CJSON_PUBLIC (cJSON *) cJSON_Parse(const char *value)
- CJSON_PUBLIC (char *) cJSON_Print(const cJSON *item)
- CJSON_PUBLIC (cJSON_bool) cJSON_PrintPreallocated(cJSON *item
- CJSON_PUBLIC (double) cJSON_GetNumberValue(const cJSON *const item)
- CJSON_PUBLIC (void *) cJSON_malloc(size_t size)

Variables

- size_t buffer_length
- const char ** return_parse_end
- const char cJSON_bool require_null_terminated
- int prebuffer
- int cJSON_bool fmt
- char * buffer
- char const int length
- char const int const cJSON_bool format
- int index
- const char *const string
- int count
- cJSON * item
- int which
- int cJSON * newitem
- cJSON *const cJSON * replacement
- cJSON_bool recurse
- const cJSON *const b
- const cJSON *const const cJSON_bool case_sensitive
- const char *const name
- const char *const const cJSON_bool boolean
- const char *const const double number
- const char *const const char *const raw
- const char * valuestring

4.9.1 Macro Definition Documentation

4.9.1.1 cJSON_Array

```
#define cJSON_Array (1 << 5)
```

Definition at line 95 of file cJSON.h.

4.9.1.2 cJSON_ArrayForEach

```
#define cJSON_ArrayForEach(
             element,
             array ) for(element = (array != NULL) ?  (array)->child :  NULL; element !=
NULL; element = element->next)
```

Definition at line 294 of file cJSON.h.

4.9.1.3 CJSON_CDECL

```
#define CJSON_CDECL
```

Definition at line 71 of file cJSON.h.

4.9.1.4 CJSON_CIRCULAR_LIMIT

```
#define CJSON_CIRCULAR_LIMIT 10000
```

Definition at line 141 of file cJSON.h.

4.9.1.5 cJSON_False

```
#define cJSON_False (1 << 0)
```

Definition at line 90 of file cJSON.h.

4.9.1.6 cJSON_Invalid

```
#define cJSON_Invalid (0)
```

Definition at line 89 of file cJSON.h.

4.9.1.7 cJSON_IsReference

```
#define cJSON_IsReference 256
```

Definition at line 99 of file cJSON.h.

4.9.1.8 CJSON_NESTING_LIMIT

```
#define CJSON_NESTING_LIMIT 1000
```

Definition at line 135 of file cJSON.h.

4.9.1.9 cJSON_NULL

```
#define cJSON_NULL (1 << 2)
```

Definition at line 92 of file cJSON.h.

4.9.1.10 cJSON_Number

```
#define cJSON_Number (1 << 3)
```

Definition at line 93 of file cJSON.h.

4.9.1.11 cJSON_Object

```
#define cJSON_Object (1 << 6)
```

Definition at line 96 of file cJSON.h.

4.9.1.12 CJSON_PUBLIC

```
#define CJSON_PUBLIC(
              type ) type
```

Definition at line 77 of file cJSON.h.

4.9.1.13 cJSON_Raw

```
#define cJSON_Raw (1 << 7) /* raw json */
```

Definition at line 97 of file cJSON.h.

4.9.1.14 cJSON_SetBoolValue

```
#define cJSON_SetBoolValue(
              object,
              boolValue )
```

Value:

```
    ( \
    (object != NULL && ((object)->type & (cJSON_False|cJSON_True))) ?  \
    (object)->type=((object)->type &(~(cJSON_False|cJSON_True)))|((boolValue)?cJSON_True:cJSON_False) :  \
    cJSON_Invalid\
)
```

Definition at line 287 of file cJSON.h.

4.9.1.15 cJSON_SetIntValue

```
#define cJSON_SetIntValue(
              object,
              number ) ((object) ?  (object)->valueint = (object)->valuedouble = (number) ↩
:  (number))
```

Definition at line 279 of file cJSON.h.

4.9.1.16 cJSON_SetNumberValue

```
#define cJSON_SetNumberValue(
              object,
              number ) ((object != NULL) ?  cJSON_SetNumberHelper(object, (double)number) ↩
:  (number))
```

Definition at line 282 of file cJSON.h.

4.9.1.17 CJSON_STDCALL

```
#define CJSON_STDCALL
```

Definition at line 72 of file cJSON.h.

4.9.1.18 cJSON_String

```
#define cJSON_String (1 << 4)
```

Definition at line 94 of file cJSON.h.

4.9.1.19 cJSON_StringIsConst

```
#define cJSON_StringIsConst 512
```

Definition at line 100 of file cJSON.h.

4.9.1.20 cJSON_True

```
#define cJSON_True (1 << 1)
```

Definition at line 91 of file cJSON.h.

4.9.1.21 CJSON_VERSION_MAJOR

```
#define CJSON_VERSION_MAJOR 1
```

Definition at line 82 of file cJSON.h.

4.9.1.22 CJSON_VERSION_MINOR

```
#define CJSON_VERSION_MINOR 7
```

Definition at line 83 of file cJSON.h.

4.9.1.23 CJSON_VERSION_PATCH

```
#define CJSON_VERSION_PATCH 18
```

Definition at line 84 of file cJSON.h.

4.9.2 Typedef Documentation

4.9.2.1 cJSON

```
typedef struct cJSON cJSON
```

4.9.2.2 cJSON_bool

```
typedef int cJSON_bool
```

Definition at line 130 of file cJSON.h.

4.9.2.3 cJSON_Hooks

```
typedef struct cJSON_Hooks cJSON_Hooks
```

4.9.3 Function Documentation

4.9.3.1 CJSON_PUBLIC() [1/7]

```
CJSON_PUBLIC (
             char *  ) const
```

4.9.3.2 CJSON_PUBLIC() [2/7]

```
CJSON_PUBLIC (
             cJSON *  ) const
```

4.9.3.3 CJSON_PUBLIC() [3/7]

```
CJSON_PUBLIC (
             cJSON_bool  )
```

4.9.3.4 CJSON_PUBLIC() [4/7]

```
CJSON_PUBLIC (
             const char *  )
```

Definition at line 94 of file cJSON.c.

4.9.3.5 CJSON_PUBLIC() [5/7]

```
CJSON_PUBLIC (
             double  ) const
```

4.9.3.6 CJSON_PUBLIC() [6/7]

```
CJSON_PUBLIC (
             void *  )
```

Definition at line 2668 of file cJSON.c.

4.9.3.7 CJSON_PUBLIC() [7/7]

```
CJSON_PUBLIC (
             void  )
```

4.9.4 Variable Documentation

4.9.4.1 b

```
const cJSON* const b
```

Definition at line 259 of file cJSON.h.

4.9.4.2 boolean

```
const char* const const cJSON_bool boolean
```

Definition at line 271 of file cJSON.h.

4.9.4.3 buffer

```
char* buffer
```

Definition at line 167 of file cJSON.h.

4.9.4.4 buffer_length

```
size_t buffer_length
```

Definition at line 153 of file cJSON.h.

4.9.4.5 case_sensitive

```
const cJSON* const const cJSON_bool case_sensitive
```

Definition at line 259 of file cJSON.h.

4.9.4.6 count

```
int count
```

Definition at line 220 of file cJSON.h.

4.9.4.7 fmt

```
int cJSON_bool fmt
```

Definition at line 164 of file cJSON.h.

4.9.4.8 format

```
char const int const cJSON_bool format
```

Definition at line 167 of file cJSON.h.

4.9.4.9 index

```
int index
```

Definition at line 174 of file cJSON.h.

4.9.4.10 item

```
cJSON *const item
```

Definition at line 226 of file cJSON.h.

4.9.4.11 length

```
char const int length
```

Definition at line 167 of file cJSON.h.

4.9.4.12 name

```
const char *const name
```

Definition at line 268 of file cJSON.h.

4.9.4.13 newitem

```
const char cJSON * newitem
```

Definition at line 246 of file cJSON.h.

4.9.4.14 number

```
double number
```

Definition at line 272 of file cJSON.h.

4.9.4.15 prebuffer

```
int prebuffer
```

Definition at line 164 of file cJSON.h.

4.9.4.16 raw

```
const char* const const char* const raw
```

Definition at line 274 of file cJSON.h.

4.9.4.17 recurse

```
cJSON_bool recurse
```

Definition at line 253 of file cJSON.h.

4.9.4.18 replacement

```
cJSON* const cJSON* replacement
```

Definition at line 247 of file cJSON.h.

4.9.4.19 require_null_terminated

```
size_t const char cJSON_bool require_null_terminated
```

Definition at line 156 of file cJSON.h.

4.9.4.20 return_parse_end

```
size_t const char ** return_parse_end
```

Definition at line 156 of file cJSON.h.

4.9.4.21 string

```
const char *const const char *const string
```

Definition at line 176 of file cJSON.h.

4.9.4.22 valuestring

```
const char* valuestring
```

Definition at line 284 of file cJSON.h.

4.9.4.23 which

```
int which
```

Definition at line 238 of file cJSON.h.

4.10 cJSON.h

Go to the documentation of this file.

```
00001 /*
00002 Copyright (c) 2009-2017 Dave Gamble and cJSON contributors
00003
00004 Permission is hereby granted, free of charge, to any person obtaining a copy
00005 of this software and associated documentation files (the "Software"), to deal
00006 in the Software without restriction, including without limitation the rights
00007 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00008 copies of the Software, and to permit persons to whom the Software is
00009 furnished to do so, subject to the following conditions:
00010
00011 The above copyright notice and this permission notice shall be included in
00012 all copies or substantial portions of the Software.
00013
00014 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00015 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00016 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00017 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00018 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00019 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
00020 THE SOFTWARE.
00021 */
00022
00023 #ifndef cJSON__h
00024 #define cJSON__h
00025
00026 #ifdef __cplusplus
00027 extern "C"
00028 {
00029 #endif
00030
00031 #if !defined(__WINDOWS__) && (defined(WIN32) || defined(WIN64) || defined(_MSC_VER) ||
      defined(_WIN32))
00032 #define __WINDOWS__
00033 #endif
00034
00035 #ifdef __WINDOWS__
00036
00037 /* When compiling for windows, we specify a specific calling convention to avoid issues where we are
      being called from a project with a different default calling convention.    For windows you have 3
      define options:
00038
00039 CJSON_HIDE_SYMBOLS - Define this in the case where you don't want to ever dllexport symbols
00040 CJSON_EXPORT_SYMBOLS - Define this on library build when you want to dllexport symbols (default)
00041 CJSON_IMPORT_SYMBOLS - Define this if you want to dllimport symbol
00042
00043 For *nix builds that support visibility attribute, you can define similar behavior by
00044
00045 setting default visibility to hidden by adding
00046 -fvisibility=hidden (for gcc)
00047 or
00048 -xldscope=hidden (for sun cc)
00049 to CFLAGS
00050
00051 then using the CJSON_API_VISIBILITY flag to "export" the same symbols the way CJSON_EXPORT_SYMBOLS
      does
00052
00053 */
00054
00055 #define CJSON_CDECL __cdecl
00056 #define CJSON_STDCALL __stdcall
00057
00058 /* export symbols by default, this is necessary for copy pasting the C and header file */
00059 #if !defined(CJSON_HIDE_SYMBOLS) && !defined(CJSON_IMPORT_SYMBOLS) && !defined(CJSON_EXPORT_SYMBOLS)
00060 #define CJSON_EXPORT_SYMBOLS
00061 #endif
00062
00063 #if defined(CJSON_HIDE_SYMBOLS)
00064 #define CJSON_PUBLIC(type)   type CJSON_STDCALL
00065 #elif defined(CJSON_EXPORT_SYMBOLS)
```

```
00066 #define CJSON_PUBLIC(type)   __declspec(dllexport) type CJSON_STDCALL
00067 #elif defined(CJSON_IMPORT_SYMBOLS)
00068 #define CJSON_PUBLIC(type)   __declspec(dllimport) type CJSON_STDCALL
00069 #endif
00070 #else /* !__WINDOWS__ */
00071 #define CJSON_CDECL
00072 #define CJSON_STDCALL
00073
00074 #if (defined(__GNUC__) || defined(__SUNPRO_CC) || defined (__SUNPRO_C)) &&
      defined(CJSON_API_VISIBILITY)
00075 #define CJSON_PUBLIC(type)   __attribute__((visibility("default"))) type
00076 #else
00077 #define CJSON_PUBLIC(type) type
00078 #endif
00079 #endif
00080
00081 /* project version */
00082 #define CJSON_VERSION_MAJOR 1
00083 #define CJSON_VERSION_MINOR 7
00084 #define CJSON_VERSION_PATCH 18
00085
00086 #include <stddef.h>
00087
00088 /* cJSON Types:  */
00089 #define cJSON_Invalid (0)
00090 #define cJSON_False  (1 << 0)
00091 #define cJSON_True   (1 << 1)
00092 #define cJSON_NULL   (1 << 2)
00093 #define cJSON_Number (1 << 3)
00094 #define cJSON_String (1 << 4)
00095 #define cJSON_Array  (1 << 5)
00096 #define cJSON_Object (1 << 6)
00097 #define cJSON_Raw    (1 << 7) /* raw json */
00098
00099 #define cJSON_IsReference 256
00100 #define cJSON_StringIsConst 512
00101
00102 /* The cJSON structure:  */
00103 typedef struct cJSON {
00104     /* next/prev allow you to walk array/object chains.  Alternatively, use
      GetArraySize/GetArrayItem/GetObjectItem */
00105     struct cJSON *next;
00106     struct cJSON *prev;
00107     /* An array or object item will have a child pointer pointing to a chain of the items in the
      array/object.  */
00108     struct cJSON *child;
00109
00110     /* The type of the item, as above.  */
00111     int type;
00112
00113     /* The item's string, if type==cJSON_String  and type == cJSON_Raw */
00114     char *valuestring;
00115     /* writing to valueint is DEPRECATED, use cJSON_SetNumberValue instead */
00116     int valueint;
00117     /* The item's number, if type==cJSON_Number */
00118     double valuedouble;
00119
00120     /* The item's name string, if this item is the child of, or is in the list of subitems of an
      object.  */
00121     char *string;
00122 } cJSON;
00123
00124 typedef struct cJSON_Hooks {
00125     /* malloc/free are CDECL on Windows regardless of the default calling convention of the compiler,
      so ensure the hooks allow passing those functions directly.  */
00126     void *(CJSON_CDECL *malloc_fn)(size_t sz);
00127     void (CJSON_CDECL *free_fn)(void *ptr);
00128 } cJSON_Hooks;
00129
00130 typedef int cJSON_bool;
00131
00132 /* Limits how deeply nested arrays/objects can be before cJSON rejects to parse them.
00133 * This is to prevent stack overflows.  */
00134 #ifndef CJSON_NESTING_LIMIT
00135 #define CJSON_NESTING_LIMIT 1000
00136 #endif
00137
00138 /* Limits the length of circular references can be before cJSON rejects to parse them.
00139 * This is to prevent stack overflows.  */
00140 #ifndef CJSON_CIRCULAR_LIMIT
00141 #define CJSON_CIRCULAR_LIMIT 10000
00142 #endif
00143
00144 /* returns the version of cJSON as a string */
00145 CJSON_PUBLIC(const char*) cJSON_Version(void);
00146
00147 /* Supply malloc, realloc and free functions to cJSON */
```

```
00148 CJSON_PUBLIC(void) cJSON_InitHooks(cJSON_Hooks* hooks);
00149
00150 /* Memory Management:  the caller is always responsible to free the results from all variants of
      cJSON_Parse (with cJSON_Delete) and cJSON_Print (with stdlib free, cJSON_Hooks.free_fn, or cJSON_free
      as appropriate).  The exception is cJSON_PrintPreallocated, where the caller has full responsibility
      of the buffer.  */
00151 /* Supply a block of JSON, and this returns a cJSON object you can interrogate.  */
00152 CJSON_PUBLIC(cJSON *) cJSON_Parse(const char *value);
00153 CJSON_PUBLIC(cJSON *) cJSON_ParseWithLength(const char *value, size_t buffer_length);
00154 /* ParseWithOpts allows you to require (and check) that the JSON is null terminated, and to retrieve
      the pointer to the final byte parsed.  */
00155 /* If you supply a ptr in return_parse_end and parsing fails, then return_parse_end will contain a
      pointer to the error so will match cJSON_GetErrorPtr().  */
00156 CJSON_PUBLIC(cJSON *) cJSON_ParseWithOpts(const char *value, const char **return_parse_end, cJSON_bool
      require_null_terminated);
00157 CJSON_PUBLIC(cJSON *) cJSON_ParseWithLengthOpts(const char *value, size_t buffer_length, const char
      **return_parse_end, cJSON_bool require_null_terminated);
00158
00159 /* Render a cJSON entity to text for transfer/storage.  */
00160 CJSON_PUBLIC(char *) cJSON_Print(const cJSON *item);
00161 /* Render a cJSON entity to text for transfer/storage without any formatting.  */
00162 CJSON_PUBLIC(char *) cJSON_PrintUnformatted(const cJSON *item);
00163 /* Render a cJSON entity to text using a buffered strategy.  prebuffer is a guess at the final size.
      guessing well reduces reallocation.  fmt=0 gives unformatted, =1 gives formatted */
00164 CJSON_PUBLIC(char *) cJSON_PrintBuffered(const cJSON *item, int prebuffer, cJSON_bool fmt);
00165 /* Render a cJSON entity to text using a buffer already allocated in memory with given length.
      Returns 1 on success and 0 on failure.  */
00166 /* NOTE: cJSON is not always 100% accurate in estimating how much memory it will use, so to be safe
      allocate 5 bytes more than you actually need */
00167 CJSON_PUBLIC(cJSON_bool) cJSON_PrintPreallocated(cJSON *item, char *buffer, const int length, const
      cJSON_bool format);
00168 /* Delete a cJSON entity and all subentities.  */
00169 CJSON_PUBLIC(void) cJSON_Delete(cJSON *item);
00170
00171 /* Returns the number of items in an array (or object).  */
00172 CJSON_PUBLIC(int) cJSON_GetArraySize(const cJSON *array);
00173 /* Retrieve item number "index" from array "array".  Returns NULL if unsuccessful.  */
00174 CJSON_PUBLIC(cJSON *) cJSON_GetArrayItem(const cJSON *array, int index);
00175 /* Get item "string" from object.  Case insensitive.  */
00176 CJSON_PUBLIC(cJSON *) cJSON_GetObjectItem(const cJSON * const object, const char * const string);
00177 CJSON_PUBLIC(cJSON *) cJSON_GetObjectItemCaseSensitive(const cJSON * const object, const char * const
      string);
00178 CJSON_PUBLIC(cJSON_bool) cJSON_HasObjectItem(const cJSON *object, const char *string);
00179 /* For analysing failed parses.  This returns a pointer to the parse error.  You'll probably need to
      look a few chars back to make sense of it.  Defined when cJSON_Parse() returns 0.  0 when
      cJSON_Parse() succeeds.  */
00180 CJSON_PUBLIC(const char *) cJSON_GetErrorPtr(void);
00181
00182 /* Check item type and return its value */
00183 CJSON_PUBLIC(char *) cJSON_GetStringValue(const cJSON * const item);
00184 CJSON_PUBLIC(double) cJSON_GetNumberValue(const cJSON * const item);
00185
00186 /* These functions check the type of an item */
00187 CJSON_PUBLIC(cJSON_bool) cJSON_IsInvalid(const cJSON * const item);
00188 CJSON_PUBLIC(cJSON_bool) cJSON_IsFalse(const cJSON * const item);
00189 CJSON_PUBLIC(cJSON_bool) cJSON_IsTrue(const cJSON * const item);
00190 CJSON_PUBLIC(cJSON_bool) cJSON_IsBool(const cJSON * const item);
00191 CJSON_PUBLIC(cJSON_bool) cJSON_IsNull(const cJSON * const item);
00192 CJSON_PUBLIC(cJSON_bool) cJSON_IsNumber(const cJSON * const item);
00193 CJSON_PUBLIC(cJSON_bool) cJSON_IsString(const cJSON * const item);
00194 CJSON_PUBLIC(cJSON_bool) cJSON_IsArray(const cJSON * const item);
00195 CJSON_PUBLIC(cJSON_bool) cJSON_IsObject(const cJSON * const item);
00196 CJSON_PUBLIC(cJSON_bool) cJSON_IsRaw(const cJSON * const item);
00197
00198 /* These calls create a cJSON item of the appropriate type.  */
00199 CJSON_PUBLIC(cJSON *) cJSON_CreateNull(void);
00200 CJSON_PUBLIC(cJSON *) cJSON_CreateTrue(void);
00201 CJSON_PUBLIC(cJSON *) cJSON_CreateFalse(void);
00202 CJSON_PUBLIC(cJSON *) cJSON_CreateBool(cJSON_bool boolean);
00203 CJSON_PUBLIC(cJSON *) cJSON_CreateNumber(double num);
00204 CJSON_PUBLIC(cJSON *) cJSON_CreateString(const char *string);
00205 /* raw json */
00206 CJSON_PUBLIC(cJSON *) cJSON_CreateRaw(const char *raw);
00207 CJSON_PUBLIC(cJSON *) cJSON_CreateArray(void);
00208 CJSON_PUBLIC(cJSON *) cJSON_CreateObject(void);
00209
00210 /* Create a string where valuestring references a string so
00211 * it will not be freed by cJSON_Delete */
00212 CJSON_PUBLIC(cJSON *) cJSON_CreateStringReference(const char *string);
00213 /* Create an object/array that only references it's elements so
00214 * they will not be freed by cJSON_Delete */
00215 CJSON_PUBLIC(cJSON *) cJSON_CreateObjectReference(const cJSON *child);
00216 CJSON_PUBLIC(cJSON *) cJSON_CreateArrayReference(const cJSON *child);
00217
00218 /* These utilities create an Array of count items.
00219 * The parameter count cannot be greater than the number of elements in the number array, otherwise
      array access will be out of bounds.*/
```

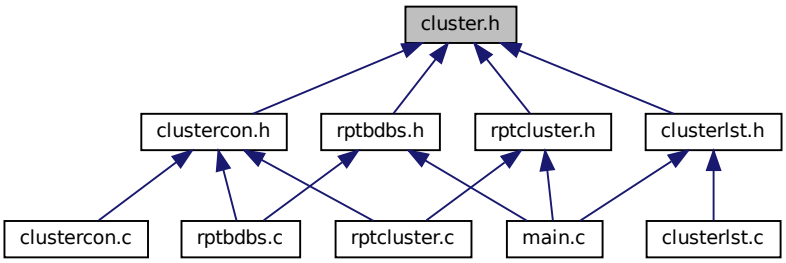
```
00220 CJSON_PUBLIC(cJSON *) cJSON_CreateIntArray(const int *numbers, int count);
00221 CJSON_PUBLIC(cJSON *) cJSON_CreateFloatArray(const float *numbers, int count);
00222 CJSON_PUBLIC(cJSON *) cJSON_CreateDoubleArray(const double *numbers, int count);
00223 CJSON_PUBLIC(cJSON *) cJSON_CreateStringArray(const char *const *strings, int count);
00224
00225 /* Append item to the specified array/object.  */
00226 CJSON_PUBLIC(cJSON_bool) cJSON_AddItemToArray(cJSON *array, cJSON *item);
00227 CJSON_PUBLIC(cJSON_bool) cJSON_AddItemToObject(cJSON *object, const char *string, cJSON *item);
00228 /* Use this when string is definitely const (i.e.  a literal, or as good as), and will definitely
     survive the cJSON object.
00229 * WARNING: When this function was used, make sure to always check that (item->type &
     cJSON_StringIsConst) is zero before
00230 * writing to `item->string` */
00231 CJSON_PUBLIC(cJSON_bool) cJSON_AddItemToObjectCS(cJSON *object, const char *string, cJSON *item);
00232 /* Append reference to item to the specified array/object.  Use this when you want to add an existing
     cJSON to a new cJSON, but don't want to corrupt your existing cJSON. */
00233 CJSON_PUBLIC(cJSON_bool) cJSON_AddItemReferenceToArray(cJSON *array, cJSON *item);
00234 CJSON_PUBLIC(cJSON_bool) cJSON_AddItemReferenceToObject(cJSON *object, const char *string, cJSON
     *item);
00235
00236 /* Remove/Detach items from Arrays/Objects.  */
00237 CJSON_PUBLIC(cJSON *) cJSON_DetachItemViaPointer(cJSON *parent, cJSON * const item);
00238 CJSON_PUBLIC(cJSON *) cJSON_DetachItemFromArray(cJSON *array, int which);
00239 CJSON_PUBLIC(void) cJSON_DeleteItemFromArray(cJSON *array, int which);
00240 CJSON_PUBLIC(cJSON *) cJSON_DetachItemFromObject(cJSON *object, const char *string);
00241 CJSON_PUBLIC(cJSON *) cJSON_DetachItemFromObjectCaseSensitive(cJSON *object, const char *string);
00242 CJSON_PUBLIC(void) cJSON_DeleteItemFromObject(cJSON *object, const char *string);
00243 CJSON_PUBLIC(void) cJSON_DeleteItemFromObjectCaseSensitive(cJSON *object, const char *string);
00244
00245 /* Update array items.  */
00246 CJSON_PUBLIC(cJSON_bool) cJSON_InsertItemInArray(cJSON *array, int which, cJSON *newitem); /* Shifts
     pre-existing items to the right.  */
00247 CJSON_PUBLIC(cJSON_bool) cJSON_ReplaceItemViaPointer(cJSON * const parent, cJSON * const item, cJSON *
     replacement);
00248 CJSON_PUBLIC(cJSON_bool) cJSON_ReplaceItemInArray(cJSON *array, int which, cJSON *newitem);
00249 CJSON_PUBLIC(cJSON_bool) cJSON_ReplaceItemInObject(cJSON *object,const char *string,cJSON *newitem);
00250 CJSON_PUBLIC(cJSON_bool) cJSON_ReplaceItemInObjectCaseSensitive(cJSON *object,const char *string,cJSON
     *newitem);
00251
00252 /* Duplicate a cJSON item */
00253 CJSON_PUBLIC(cJSON *) cJSON_Duplicate(const cJSON *item, cJSON_bool recurse);
00254 /* Duplicate will create a new, identical cJSON item to the one you pass, in new memory that will
00255 * need to be released.  With recurse!=0, it will duplicate any children connected to the item.
00256 * The item->next and ->prev pointers are always zero on return from Duplicate.  */
00257 /* Recursively compare two cJSON items for equality.  If either a or b is NULL or invalid, they will
     be considered unequal.
00258 * case_sensitive determines if object keys are treated case sensitive (1) or case insensitive (0) */
00259 CJSON_PUBLIC(cJSON_bool) cJSON_Compare(const cJSON * const a, const cJSON * const b, const cJSON_bool
     case_sensitive);
00260
00261 /* Minify a strings, remove blank characters(such as ' ', '\t', '\r', '\n') from strings.
00262 * The input pointer json cannot point to a read-only address area, such as a string constant,
00263 * but should point to a readable and writable address area.  */
00264 CJSON_PUBLIC(void) cJSON_Minify(char *json);
00265
00266 /* Helper functions for creating and adding items to an object at the same time.
00267 * They return the added item or NULL on failure.  */
00268 CJSON_PUBLIC(cJSON*) cJSON_AddNullToObject(cJSON * const object, const char * const name);
00269 CJSON_PUBLIC(cJSON*) cJSON_AddTrueToObject(cJSON * const object, const char * const name);
00270 CJSON_PUBLIC(cJSON*) cJSON_AddFalseToObject(cJSON * const object, const char * const name);
00271 CJSON_PUBLIC(cJSON*) cJSON_AddBoolToObject(cJSON * const object, const char * const name, const
     cJSON_bool boolean);
00272 CJSON_PUBLIC(cJSON*) cJSON_AddNumberToObject(cJSON * const object, const char * const name, const
     double number);
00273 CJSON_PUBLIC(cJSON*) cJSON_AddStringToObject(cJSON * const object, const char * const name, const char
     * const string);
00274 CJSON_PUBLIC(cJSON*) cJSON_AddRawToObject(cJSON * const object, const char * const name, const char *
     const raw);
00275 CJSON_PUBLIC(cJSON*) cJSON_AddObjectToObject(cJSON * const object, const char * const name);
00276 CJSON_PUBLIC(cJSON*) cJSON_AddArrayToObject(cJSON * const object, const char * const name);
00277
00278 /* When assigning an integer value, it needs to be propagated to valuedouble too.  */
00279 #define cJSON_SetIntValue(object, number) ((object) ?  (object)->valueint = (object)->valuedouble =
     (number) :  (number))
00280 /* helper for the cJSON_SetNumberValue macro */
00281 CJSON_PUBLIC(double) cJSON_SetNumberHelper(cJSON *object, double number);
00282 #define cJSON_SetNumberValue(object, number) ((object != NULL) ? cJSON_SetNumberHelper(object,
     (double)number) :  (number))
00283 /* Change the valuestring of a cJSON_String object, only takes effect when type of object is
     cJSON_String */
00284 CJSON_PUBLIC(char*) cJSON_SetValuestring(cJSON *object, const char *valuestring);
00285
00286 /* If the object is not a boolean type this does nothing and returns cJSON_Invalid else it returns the
     new type*/
00287 #define cJSON_SetBoolValue(object, boolValue) ( \
00288 (object != NULL && ((object)->type & (cJSON_False|cJSON_True))) ?  \
00289 (object)->type=((object)->type &(~(cJSON_False|cJSON_True)))|((boolValue)?cJSON_True:cJSON_False) :  \
```

```
00290 cJSON_Invalid\
00291 )
00292
00293 /* Macro for iterating over an array or object */
00294 #define cJSON_ArrayForEach(element, array) for(element = (array != NULL) ? (array)->child :  NULL;
      element != NULL; element = element->next)
00295
00296 /* malloc/free objects using the malloc/free functions that have been set with cJSON_InitHooks */
00297 CJSON_PUBLIC(void *) cJSON_malloc(size_t size);
00298 CJSON_PUBLIC(void) cJSON_free(void *object);
00299
00300 #ifdef __cplusplus
00301 }
00302 #endif
00303
00304 #endif
```

4.11 cluster.h File Reference

<+DETAILED+>

This graph shows which files directly or indirectly include this file:



Classes

- struct cluster_s

Typedefs

- typedef struct cluster_s cluster_t

4.11.1 Detailed Description

<+DETAILED+>

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file cluster.h.

4.11.2 Typedef Documentation

4.11.2.1 cluster_t

```
typedef struct cluster_s cluster_t
```

4.12 cluster.h

Go to the documentation of this file.

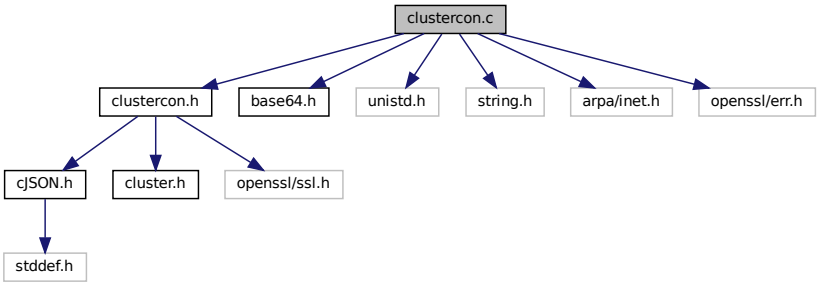
```
00001
00021 #ifndef __CLUSTER_H__
00022 #define __CLUSTER_H__
00023
00024 typedef struct cluster_s {
00025     unsigned short int enabled;
00026     char* host;
00027     char* user;
00028     char* pass;
00029     char* insecure;
00030     char* cacert;
00031 } cluster_t;
00032
00033 #endif /* __CLUSTER_H__ */
00034 /* vim:  set tw=80:  */
```

4.13 clustercon.c File Reference

<+DETAILED+>

```
#include "clustercon.h"
#include "base64.h"
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <openssl/err.h>
```

Include dependency graph for clustercon.c:



Functions

- rsclustercon_t * cluster_new (const cluster_t *cluster)
- int cluster_open (rsclustercon_t *rsclustercon)
- cJSON * cluster_queryget (const rsclustercon_t *rsclustercon, const char *endpoint)
- void cluster_close (rsclustercon_t *rsclustercon)
- void cluster_del (rsclustercon_t *rsclustercon)

4.13.1 Detailed Description

<+DETAILED+>

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file clustercon.c.

4.13.2 Function Documentation

4.13.2.1 cluster_close()

```
void cluster_close (
            rsclustercon_t * rsclustercon )
```

Definition at line 206 of file clustercon.c.

4.13.2.2 cluster_del()

```
void cluster_del (
            rsclustercon_t * rsclustercon )
```

Definition at line 214 of file clustercon.c.

4.13.2.3 cluster_new()

```
rsclustercon_t * cluster_new (
            const cluster_t * cluster )
```

Definition at line 38 of file clustercon.c.

4.13.2.4 cluster_open()

```
int cluster_open (
            rsclustercon_t * rsclustercon )
```

Definition at line 75 of file clustercon.c.

4.13.2.5 cluster_queryget()

```
cJSON * cluster_queryget (
            const rsclustercon_t * rsclustercon,
            const char * endpoint )
```

Definition at line 139 of file clustercon.c.

Here is the call graph for this function:

4.14 clustercon.c

Go to the documentation of this file.

```
00001
00021 #ifdef HAVE_CONFIG_H
00022 #include "config.h"
00023 #endif
00024
00025 #include "clustercon.h"
00026 #include "base64.h"                          /* Base64 encoder and decoder */
00027
00028 #include <unistd.h>
00029 #include <string.h>
00030 #ifdef _WIN32
00031 #include <winsock2.h>
00032 #include <ws2tcpip.h>
00033 #else
00034 #include <arpa/inet.h>
00035 #endif
00036 #include <openssl/err.h>
00037
00038 rsclustercon_t*    cluster_new(const cluster_t* cluster) {
00039     rsclustercon_t* rsclustercon = NULL;
00040
00041     if (NULL==(rsclustercon=malloc(sizeof(struct rsclustercon_s)))) {
00042         perror("cluster_new");
00043         return NULL;
00044     }
00045     if (NULL==(rsclustercon->host=strdup(cluster->host))) {
00046         perror("cluster_new host");
00047         free(rsclustercon);
00048         return NULL;
00049     }
00050     if (NULL==(rsclustercon->user=strdup(cluster->user))) {
00051         perror("cluster_new user");
```

```
00052         free(rsclustercon->host);
00053         free(rsclustercon);
00054         return NULL;
00055     }
00056     if (NULL==(rsclustercon->pass=strdup(cluster->pass))) {
00057         perror("cluster_new pass");
00058         free(rsclustercon->user);
00059         free(rsclustercon->host);
00060         free(rsclustercon);
00061         return NULL;
00062     }
00063     if (NULL==(rsclustercon->cacert=strdup(cluster->cacert))) {
00064         perror("cluster_new cacert");
00065         free(rsclustercon->pass);
00066         free(rsclustercon->user);
00067         free(rsclustercon->host);
00068         free(rsclustercon);
00069         return NULL;
00070     }
00071     rsclustercon->insecure = strcmp(cluster->insecure,"false");
00072     return rsclustercon;
00073 }
00074
00075 int   cluster_open(rsclustercon_t* rsclustercon) {
00076     struct sockaddr_in server_addr;
00077
00078     /* Socket creation */
00079     rsclustercon->sock = socket(AF_INET, SOCK_STREAM, 0);
00080     if (rsclustercon->sock < 0) {
00081         perror("Socket creation error");
00082         return 1;
00083     }
00084
00085     server_addr.sin_family = AF_INET;
00086     server_addr.sin_port = htons(9443);
00087     if (inet_pton(AF_INET, rsclustercon->host, &server_addr.sin_addr) <= 0) {
00088         perror("Invalid address");
00089         return 2;
00090     }
00091
00092     /* Server connection */
00093     if (connect(rsclustercon->sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) != 0) {
00094         perror("Connection failed");
00095         return 3;
00096     }
00097
00098     /* OpenSSL initialization */
00099     SSL_library_init();
00100     /* OpenSSL_add_all_algorithms(); */
00101     OpenSSL_add_ssl_algorithms();
00102     SSL_load_error_strings();
00103
00104     /* Initialize OpenSSL context */
00105     rsclustercon->ctx = SSL_CTX_new(SSLv23_client_method());
00106     if (!rsclustercon->ctx) {
00107         ERR_print_errors_fp(stderr);
00108         return 4;
00109     }
00110
00111     /* Load server certificate or certificate from a certification authority (CA) */
00112     if (strcmp("",rsclustercon->cacert)&&rsclustercon->insecure==0)
00113         if (SSL_CTX_load_verify_locations(rsclustercon->ctx, rsclustercon->cacert, NULL) != 1) {
00114             ERR_print_errors_fp(stderr);
00115             return 5;
00116         }
00117
00118     if (rsclustercon->insecure)
00119         /* Disable certificat check */
00120         SSL_CTX_set_verify(rsclustercon->ctx, SSL_VERIFY_NONE, NULL);
00121     else
00122         /* Enable certificat check */
00123         SSL_CTX_set_verify(rsclustercon->ctx, SSL_VERIFY_PEER, NULL);
00124
00125
00126     /* Link SSL configuration to the socket */
00127     rsclustercon->ssl = SSL_new(rsclustercon->ctx);
00128     SSL_set_fd(rsclustercon->ssl, rsclustercon->sock);
00129
00130     /* Initiate SSL connection */
00131     if (SSL_connect(rsclustercon->ssl) <= 0) {
00132         ERR_print_errors_fp(stderr);
00133         return 6;
00134     }
00135
00136     return 0;
00137 }
00138
```

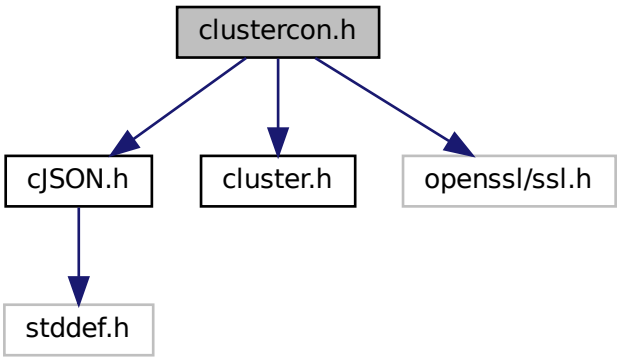
```
00139 cJSON* cluster_queryget(const rsclustercon_t* rsclustercon, const char* endpoint) {
00140     char buf[1024];
00141 
00142     /* Prepare basic authentication */
00143     char* auth_encoded;
00144     {
00145         char* auth_clear;
00146         if (NULL==(auth_clear =
      (char*)malloc(strlen(rsclustercon->user)+1+strlen(rsclustercon->pass)+1))) {
00147             perror("cluster_queryget malloc(auth)");
00148             return NULL;
00149         };
00150         strcpy(auth_clear,rsclustercon->user);
00151         strcat(auth_clear,":");
00152         strcat(auth_clear,rsclustercon->pass);
00153         auth_encoded=base64_encode(auth_clear);
00154         free(auth_clear);
00155     }
00156 
00157     /* Prepare query */
00158     char http_request[512];
00159     snprintf(http_request, sizeof(http_request),
00160              "GET %s HTTP/1.1\r\n"
00161              "Host:  %s\r\n"
00162              "Content-Type:  application/json\r\n"
00163              "Authorization:  Basic %s\r\n"
00164              "Connection:  close\r\n\r\n",
00165              endpoint,rsclustercon->host,auth_encoded
00166             );
00167     free(auth_encoded);
00168 
00169     /* Send query */
00170     SSL_write(rsclustercon->ssl, http_request, strlen(http_request));
00171 
00172     /* Read reply */
00173     int bytes=0;
00174     char* reply=NULL;
00175     if (NULL==(reply=strdup(""))) {
00176         perror("cluster_queryget strdup");
00177         return NULL;
00178     };
00179     while ((bytes=SSL_read(rsclustercon->ssl, buf, 1024 - 1)) > 0) {
00180         buf[bytes] = 0;
00181         char* newreply;
00182         if ((newreply=(char*)realloc(reply,strlen(reply)+bytes+1))==NULL) {
00183             perror("Unable to allocate reply buffer");
00184             free(reply);
00185             return NULL;
00186         } else
00187             reply = newreply;
00188         strcat(reply, buf);
00189     }
00190 
00191     /* Remove HTTP headers */
00192     char* http_body;
00193     char* retval_txt;
00194     if (NULL==(http_body = strstr(reply,"\r\n\r\n")))
00195         retval_txt= NULL;
00196     else
00197         retval_txt=strdup(http_body);
00198     free(reply);
00199 
00200     cJSON* retval_json = cJSON_Parse(retval_txt);
00201     free(retval_txt);
00202 
00203     return retval_json;
00204 }
00205 
00206 void  cluster_close(rsclustercon_t* rsclustercon) {
00207     SSL_shutdown(rsclustercon->ssl);
00208     SSL_free(rsclustercon->ssl);
00209     close(rsclustercon->sock);
00210     SSL_CTX_free(rsclustercon->ctx);
00211     EVP_cleanup();
00212 }
00213 
00214 void  cluster_del(rsclustercon_t* rsclustercon) {
00215     free(rsclustercon->cacert);
00216     free(rsclustercon->pass);
00217     free(rsclustercon->user);
00218     free(rsclustercon->host);
00219     free(rsclustercon);
00220 }
00221 
00222 /* vim:  set tw=80:  */
```

4.15 clustercon.h File Reference

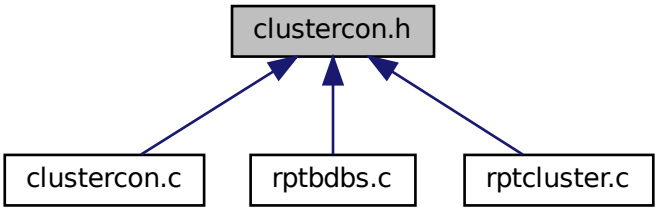
<+DETAILED+>

```
#include "cJSON.h"
#include "cluster.h"
#include <openssl/ssl.h>
```

Include dependency graph for clustercon.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct rsclustercon_s

Typedefs

- typedef struct rsclustercon_s rsclustercon_t

Functions

- rsclustercon_t * cluster_new (const cluster_t *cluster)
- int cluster_open (rsclustercon_t *rsclustercon)
- cJSON * cluster_queryget (const rsclustercon_t *rsclustercon, const char *endpoint)
- void cluster_close (rsclustercon_t *rsclustercon)
- void cluster_del (rsclustercon_t *rsclustercon)

4.15.1 Detailed Description

<+DETAILED+>

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file clustercon.h.

4.15.2 Typedef Documentation

4.15.2.1 rsclustercon_t

```
typedef struct rsclustercon_s rsclustercon_t
```

4.15.3 Function Documentation

4.15.3.1 cluster_close()

```
void cluster_close (
            rsclustercon_t * rsclustercon )
```

Definition at line 206 of file clustercon.c.

4.15.3.2 cluster_del()

```
void cluster_del (
            rsclustercon_t * rsclustercon )
```

Definition at line 214 of file clustercon.c.

4.15.3.3 cluster_new()

```
rsclustercon_t * cluster_new (
            const cluster_t * cluster )
```

Definition at line 38 of file clustercon.c.

4.15.3.4 cluster_open()

```
int cluster_open (
            rsclustercon_t * rsclustercon )
```

Definition at line 75 of file clustercon.c.

4.15.3.5 cluster_queryget()

```
cJSON * cluster_queryget (
            const rsclustercon_t * rsclustercon,
            const char * endpoint )
```

Definition at line 139 of file clustercon.c.

Here is the call graph for this function:

4.16 clustercon.h

Go to the documentation of this file.

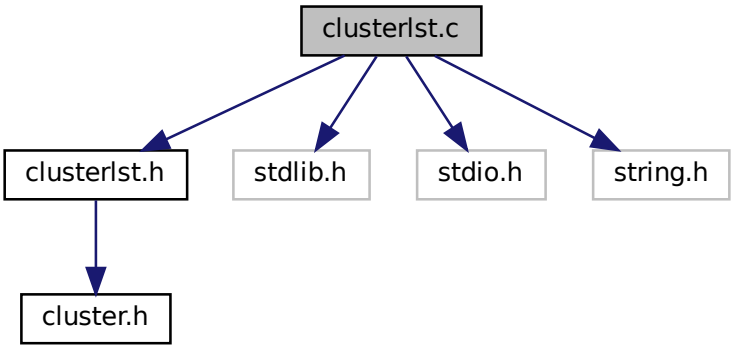
```
00001
00021 #ifndef __CLUSTERCON_H__
00022 #define __CLUSTERCON_H__
00023
00024 #include "cJSON.h"
00025 #include "cluster.h"
00026
00027 #include <openssl/ssl.h>
00028
00029 typedef struct rsclustercon_s {
00030     char*               host;
00031     char*               user;
00032     char*               pass;
00033     unsigned short int insecure;
00034     char*               cacert;
00035     int                 sock;
00036     SSL_CTX*            ctx;
00037     SSL*                ssl;
00038 } rsclustercon_t;
00039
00040 rsclustercon_t* cluster_new(const cluster_t* cluster);
00041 int             cluster_open(rsclustercon_t* rsclustercon);
00042 cJSON*          cluster_queryget(const rsclustercon_t* rsclustercon, const char* endpoint);
00043 void            cluster_close(rsclustercon_t* rsclustercon);
00044 void            cluster_del(rsclustercon_t* rsclustercon);
00045
00046 #endif /* __CLUSTERCON_H__ */
00047 /* vim:  set tw=80:  */
```

4.17 clusterlst.c File Reference

Basic(non-thread-safe) single-chained list of record with sentinal.

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

Include dependency graph for clusterlst.c:



Classes

- struct clusterrecord_s

Typedefs

- typedef struct clusterrecord_s clusterrecord_t

Variables

- void(∗ clusterlist_add)(cluster_t ∗cluster) = clusterlist_add_preinit
- cluster_t ∗(∗ clusterlist_find)(const char ∗host) = clusterlist_find_preinit
- cluster_t ∗(∗ clusterlist_first)() = clusterlist_first_preinit
- cluster_t ∗(∗ clusterlist_next)() = clusterlist_next_preinit
- cluster_t ∗(∗ clusterlist_get)() = clusterlist_get_preinit

4.17.1 Detailed Description

Basic(non-thread-safe) single-chained list of record with sentinal.

<+DETAILED+>

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file clusterlst.c.

4.17.2 Typedef Documentation

4.17.2.1 clusterrecord_t

```
typedef struct clusterrecord_s clusterrecord_t
```

4.17.3 Variable Documentation

4.17.3.1 clusterlist_add

```
void(* clusterlist_add) (cluster_t *cluster) (
            cluster_t * cluster ) = clusterlist_add_preinit
```

Definition at line 82 of file clusterlst.c.

4.17.3.2 clusterlist_find

```
cluster_t *(* clusterlist_find) (const char *host) (
            const char * host ) = clusterlist_find_preinit
```

Definition at line 95 of file clusterlst.c.

4.17.3.3 clusterlist_first

```
cluster_t *(* clusterlist_first) () ( ) = clusterlist_first_preinit
```

Definition at line 105 of file clusterlst.c.

4.17.3.4 clusterlist_get

```
cluster_t *(* clusterlist_get) () ( ) = clusterlist_get_preinit
```

Definition at line 125 of file clusterlst.c.

4.17.3.5 clusterlist_next

```
cluster_t *(* clusterlist_next) () ( ) = clusterlist_next_preinit
```

Definition at line 116 of file clusterlst.c.

4.18 clusterlst.c

Go to the documentation of this file.

```
00001
00021 #ifdef HAVE_CONFIG_H
00022 #include "config.h"
00023 #endif
00024
00025 #include "clusterlst.h"
00026
00027 #include <stdlib.h>
00028 #include <stdio.h>
00029 #include <string.h>
00030
00031 typedef struct clusterrecord_s {
00032     struct clusterrecord_s* next;
00033     cluster_t* cluster;
00034 } clusterrecord_t;
00035
00036 static clusterrecord_t* clusterlistfirst;
00037 static clusterrecord_t* clusterlistlast;
00038 static clusterrecord_t* clusterlistcursor;
00039
00040 static void clusterlist_add_postinit (cluster_t* cluster);
00041 static cluster_t* clusterlist_find_postinit (const char* host);
00042 static cluster_t* clusterlist_first_postinit();
00043 static cluster_t* clusterlist_next_postinit();
00044 static cluster_t* clusterlist_get_postinit();
00045
00046 static void clusterlist_init() {
00047     if (NULL==(clusterlistfirst=malloc(sizeof(struct clusterrecord_s)))) {
00048         perror("clusterlist_init");
00049         exit(EXIT_FAILURE);
00050     }
00051     /* Initialize chained structure */
00052     clusterlistlast = clusterlistfirst;
00053     clusterlistcursor = clusterlistfirst;
00054     clusterlistlast->next=NULL;
00055     clusterlistlast->cluster=NULL;
00056
00057     /* From now, the application can use the real functions */
00058     clusterlist_add = clusterlist_add_postinit;
00059     clusterlist_find = clusterlist_find_postinit;
00060     clusterlist_first = clusterlist_first_postinit;
00061     clusterlist_next = clusterlist_next_postinit;
00062     clusterlist_get = clusterlist_get_postinit;
00063 }
00064
00065 static void clusterlist_add_postinit (cluster_t* cluster) {
00066     /* Store the record in the sentinel */
00067     clusterlistlast->cluster=cluster;
00068
00069     /* Create a new sentinel */
00070     if (NULL==(clusterlistlast->next=malloc(sizeof(struct clusterrecord_s)))) {
00071         perror("clusterlist_add_postinit malloc(sentinel)");
00072         exit(EXIT_FAILURE);
00073     }
00074     clusterlistlast = clusterlistlast->next;
00075     clusterlistlast->next=NULL;
00076     clusterlistlast->cluster=NULL;
00077 }
00078 static void clusterlist_add_preinit (cluster_t* cluster) {
00079     clusterlist_init();
00080     clusterlist_add(cluster);
00081 }
00082 void (*clusterlist_add)(cluster_t* cluster) = clusterlist_add_preinit;
00083
00084 static cluster_t* clusterlist_find_postinit(const char* host) {
00085     clusterlistcursor = clusterlistfirst;
00086     while (clusterlistcursor->cluster&&
00087             (strcmp(host,clusterlistcursor->cluster->host)))
00088         clusterlistcursor = clusterlistcursor->next;
```

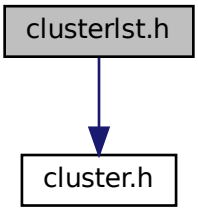
```
00089     return (clusterlistcursor->cluster);
00090 }
00091 static cluster_t* clusterlist_find_preinit(const char* host) {
00092     clusterlist_init();
00093     return clusterlist_find(host);
00094 }
00095 cluster_t* (*clusterlist_find)(const char* host) = clusterlist_find_preinit;
00096
00097 static cluster_t* clusterlist_first_postinit() {
00098     clusterlistcursor = clusterlistfirst;
00099     return (clusterlistcursor->cluster);
00100 }
00101 static cluster_t* clusterlist_first_preinit() {
00102     clusterlist_init();
00103     return clusterlist_first();
00104 }
00105 cluster_t* (*clusterlist_first)() = clusterlist_first_preinit;
00106
00107 static cluster_t* clusterlist_next_postinit() {
00108     if (clusterlistcursor->next)
00109         clusterlistcursor = clusterlistcursor->next;
00110     return (clusterlistcursor->cluster);
00111 }
00112 static cluster_t* clusterlist_next_preinit() {
00113     clusterlist_init();
00114     return clusterlist_next();
00115 }
00116 cluster_t* (*clusterlist_next)() = clusterlist_next_preinit;
00117
00118 static cluster_t* clusterlist_get_postinit() {
00119     return (clusterlistcursor->cluster);
00120 }
00121 static cluster_t* clusterlist_get_preinit() {
00122     clusterlist_init();
00123     return clusterlist_get();
00124 }
00125 cluster_t* (*clusterlist_get)() = clusterlist_get_preinit;
00126
00127 /* vim:  set tw=80:  */
```

4.19 clusterlst.h File Reference

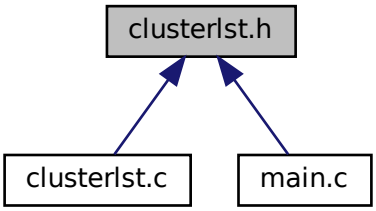
Basic(non-thread-safe) single-chained list of record with sentinal.

`#include "cluster.h"`

Include dependency graph for clusterlst.h:



This graph shows which files directly or indirectly include this file:



Variables

- void(∗ clusterlist_add)(cluster_t ∗cluster)
- cluster_t ∗(∗ clusterlist_find)(const char ∗host)
- cluster_t ∗(∗ clusterlist_first)()
- cluster_t ∗(∗ clusterlist_next)()
- cluster_t ∗(∗ clusterlist_get)()

4.19.1 Detailed Description

Basic(non-thread-safe) single-chained list of record with sentinal.

<+DETAILED+>

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file clusterlst.h.

4.19.2 Variable Documentation

4.19.2.1 clusterlist_add

```
void(* clusterlist_add) (cluster_t *cluster) (
            cluster_t * cluster )  [extern]
```

Definition at line 82 of file clusterlst.c.

4.19.2.2 clusterlist_find

```
cluster_t *(* clusterlist_find) (const char *host) (
            const char * host )  [extern]
```

Definition at line 95 of file clusterlst.c.

4.19.2.3 clusterlist_first

```
cluster_t *(* clusterlist_first) () ( )  [extern]
```

Definition at line 105 of file clusterlst.c.

4.19.2.4 clusterlist_get

```
cluster_t *(* clusterlist_get) () ( )  [extern]
```

Definition at line 125 of file clusterlst.c.

4.19.2.5 clusterlist_next

```
cluster_t *(* clusterlist_next) () ( )  [extern]
```

Definition at line 116 of file clusterlst.c.

4.20 clusterlst.h

Go to the documentation of this file.

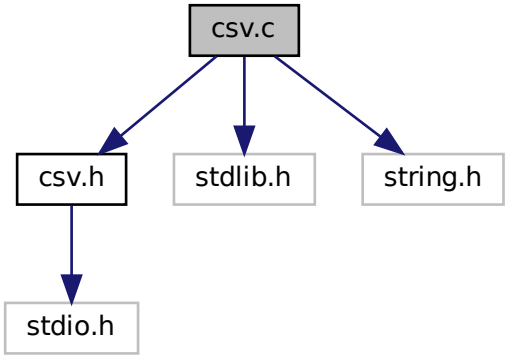
```
00001
00021 #ifndef __CLUSTERLST_H__
00022 #define __CLUSTERLST_H__
00023
00024 #include "cluster.h"
00025
00026 extern void (*clusterlist_add)(cluster_t* cluster);
00027 extern cluster_t* (*clusterlist_find) (const char* host);
00028 extern cluster_t* (*clusterlist_first)();
00029 extern cluster_t* (*clusterlist_next)();
00030 extern cluster_t* (*clusterlist_get)();
00031
00032 #endif /* __CLUSTERLST_H__ */
00033 /* vim:  set tw=80:  */
```

4.21 csv.c File Reference

https://www.rfc-editor.org/rfc/rfc4180

```
#include "csv.h"
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for csv.c:



Functions

- void csv_addline (FILE *reportfile)
- void csv_addfield (FILE *reportfile, const char *value)
- char * csvtok (char *source)
- char * txt2csv (const char *text)

4.21.1 Detailed Description

https://www.rfc-editor.org/rfc/rfc4180

<+DETAILED+>

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file csv.c.

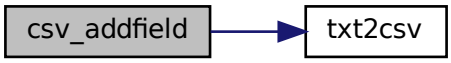
4.21.2 Function Documentation

4.21.2.1 csv_addfield()

```
void csv_addfield (
            FILE * reportfile,
            const char * value )
```

Definition at line 37 of file csv.c.

Here is the call graph for this function:



4.21.2.2 csv_addline()

```
void csv_addline (
            FILE * reportfile )
```

Definition at line 32 of file csv.c.

4.21.2.3 csvtok()

```
char * csvtok (
            char * source )
```

Definition at line 50 of file csv.c.

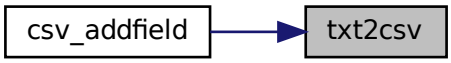
Here is the caller graph for this function:

4.21.2.4 txt2csv()

```
char * txt2csv (
            const char * text )
```

Definition at line 146 of file csv.c.

Here is the caller graph for this function:



4.22 csv.c

Go to the documentation of this file.

```
00001
00021 #ifdef HAVE_CONFIG_H
00022 #include "config.h"
00023 #endif
00024
00025 #include "csv.h"
00026
00027 #include <stdlib.h>
00028 #include <string.h>
00029
00030 static unsigned short int _csv_firstfield = 1;
00031
00032 void csv_addline(FILE* reportfile) {
00033     fprintf(reportfile,"\r\n");
00034     _csv_firstfield = 1;
00035 }
00036
00037 void csv_addfield(FILE* reportfile, const char* value) {
00038     char* csv = txt2csv(value);
00039     if (_csv_firstfield) {
00040         fprintf(reportfile,"%s",csv);
00041         _csv_firstfield = 0;
00042     } else
00043         fprintf(reportfile,",%s",csv);
00044     free(csv);
00045 }
00046
00047 /* RFC4180 compliant CSV parser, with LF only tolerance when CRLF expected */
00048 /* https://www.rfc-editor.org/rfc/rfc4180 */
00049 static char* _csvtok_csv=NULL;
00050 char* csvtok(char* source) {
00051     size_t csv_idx=0;
00052     size_t txt_idx=0;
00053     char* csvtok_txt = NULL;
00054     unsigned short int quoted = 0;
00055
00056     if (NULL!=source)
00057         _csvtok_csv = source;
00058
00059     if (_csvtok_csv[0]==0)
00060         return NULL;
00061
00062     if (NULL==(csvtok_txt = malloc(strlen(_csvtok_csv)+1))) {
00063         perror("csvtok csvtok_txt malloc");
00064         exit(EXIT_FAILURE);
00065     }
00066     csvtok_txt[0]=0;
00067
00068     while (_csvtok_csv[csv_idx]) {
```

```
00069         if (!quoted) {
00070             if (_csvtok_csv[csv_idx]==',') {
00071                 if ((_csvtok_csv[csv_idx+1]==0)||
00072                         (_csvtok_csv[csv_idx+1]=='\n')|| /* Unix style tolerance */
00073                         ((_csvtok_csv[csv_idx+1]=='\r')&&(_csvtok_csv[csv_idx+2]=='\n'))) {
00074                     fprintf(stderr,"RFC4180 forbids comma at the end of record %s at %zu.\n",
00075                             _csvtok_csv,csv_idx);
00076                     free(csvtok_txt);
00077                     _csvtok_csv=NULL;
00078                     return NULL;
00079                 } else
00080                     break;
00081             } else if ((_csvtok_csv[csv_idx]=='\n')||
00082                        ((_csvtok_csv[csv_idx]=='\r')&&(_csvtok_csv[csv_idx+1]=='\n')))
00083                 break;
00084             else if (_csvtok_csv[csv_idx]=='"') {
00085                 if (csv_idx==0) {
00086                     quoted = 1;
00087                     csv_idx++;
00088                     continue;
00089                 } else {
00090                     fprintf(stderr,"doublequote in a non quoted value %s at %zu.\n",
00091                             _csvtok_csv,csv_idx);
00092                     free(csvtok_txt);
00093                     _csvtok_csv=NULL;
00094                     return NULL;
00095                 }
00096             }
00097         } else { /* Quoted */
00098             if (_csvtok_csv[csv_idx]=='"') {
00099                 if (_csvtok_csv[csv_idx+1]=='"') {
00100                     /* Skip escaping doublequote and let the copy occur */
00101                     csv_idx++;
00102                 } else if ((_csvtok_csv[csv_idx+1]=='0')||
00103                            (_csvtok_csv[csv_idx+1]==',')||
00104                            (_csvtok_csv[csv_idx+1]=='\n')||
00105                            ((_csvtok_csv[csv_idx+1]=='\r')&&(_csvtok_csv[csv_idx+2]=='\n'))) {
00106                     quoted = 0;
00107                     csv_idx++;
00108                     break;
00109                 } else {
00110                     fprintf(stderr,"doublequote should be at the end of field or escaping another
      doublequote in %s at %zu.\n",
00111                             _csvtok_csv,csv_idx);
00112                     free(csvtok_txt);
00113                     _csvtok_csv=NULL;
00114                     return NULL;
00115                 }
00116             }
00117         }
00118         csvtok_txt[txt_idx++]=_csvtok_csv[csv_idx++];
00119     }
00120
00121     /* Close csvtok_txt */
00122     csvtok_txt[txt_idx]=0;
00123     if (quoted) {
00124         fprintf(stderr,"Missing end-of-field doublequote %s\n",csvtok_txt);
00125         free(csvtok_txt);
00126         _csvtok_csv=NULL;
00127         return NULL;
00128     } else if (_csvtok_csv[csv_idx]==',') /* Next char after end of field should be 0/,/CRLF */
00129         csv_idx++;
00130     else if (_csvtok_csv[csv_idx]=='\n')
00131         csv_idx+=1;
00132     else if ((_csvtok_csv[csv_idx]=='\r')&&(_csvtok_csv[csv_idx+1]=='\n'))
00133         csv_idx+=2;
00134     else if (_csvtok_csv[csv_idx]!=0) {
00135         fprintf(stderr,"Parsing error after %s\n",csvtok_txt);
00136         free(csvtok_txt);
00137         _csvtok_csv=NULL;
00138         return NULL;
00139     }
00140     _csvtok_csv += csv_idx;
00141     return csvtok_txt;
00142 }
00143
00144 /* RFC4180 compliant text to CSV encoder */
00145 /* https://www.rfc-editor.org/rfc/rfc4180 */
00146 char* txt2csv(const char* text) {
00147     char* csv;
00148     size_t text_idx;
00149     size_t csv_idx;
00150     unsigned short int need_quotes=0;
00151
00152     {
00153         /* Check if quotes are needed and how many doublequotes are in the
00154 * source text to allocate output buffer.  This text iteration could be
```

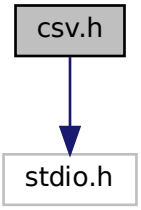
```
00155 * avoided but would imply to overallocate for the worst case scenario
00156 * and to reallocate at the end with the potentially needed surrounding
00157 * quotes */
00158         size_t extra_chars = 0;
00159         for (text_idx=0; text[text_idx]; text_idx++) {
00160             if ( (text[text_idx]==',')|| (text[text_idx]=='\r')|| (text[text_idx]=='\n'))
00161                 need_quotes = 1;
00162             if (text[text_idx]=='"') {
00163                 need_quotes = 1;
00164                 extra_chars++;
00165             }
00166         }
00167         /* Allocate the right output buffer size */
00168         if (NULL==(csv=malloc(strlen(text)+(need_quotes?2:0)+extra_chars+1))) {
00169             perror("txt2csv malloc");
00170             return NULL;
00171         }
00172     }
00173 
00174     text_idx = 0;
00175     csv_idx = 0;
00176 
00177     /* If quotes are needed add a starting doublequote */
00178     if (need_quotes)
00179         csv[csv_idx++] = '"';
00180 
00181     /* Copy each source char to the destination buffer */
00182     while (text[text_idx]) {
00183         /* With a doublequote before if the char to copy is a doublequote */
00184         if (text[text_idx]=='"')
00185             csv[csv_idx++] = '"';
00186         csv[csv_idx++] = text[text_idx++];
00187     }
00188 
00189     /* If quotes are needed add a closing doublequote */
00190     if (need_quotes)
00191         csv[csv_idx++] = '"';
00192 
00193     /* Properly end the C string */
00194     csv[csv_idx] = 0;
00195 
00196     return csv;
00197 }
00198 /* vim:  set tw=80:  */
```

4.23 csv.h File Reference

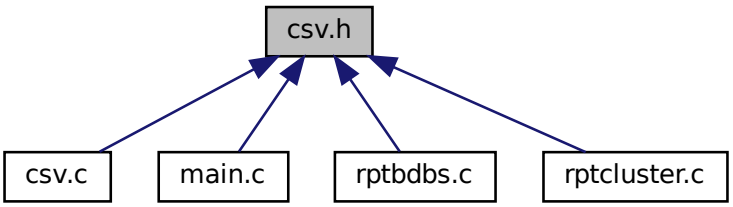
<+DETAILED+>

`#include <stdio.h>`

Include dependency graph for csv.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef struct csv_s csv_t
- typedef struct csvrecord_s csvrecord_t
- typedef struct csvfield_s csvfield_t

Functions

- void csv_addline (FILE ∗reportfile)
- void csv_addfield (FILE ∗reportfile, const char ∗value)
- char ∗ csvtok (char ∗source)
- char ∗ txt2csv (const char ∗text)

4.23.1 Detailed Description

<+DETAILED+>

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file csv.h.

4.23.2 Typedef Documentation

4.23.2.1 csv_t

```
typedef struct csv_s csv_t
```

Definition at line 26 of file csv.h.

4.23.2.2 csvfield_t

```
typedef struct csvfield_s csvfield_t
```

Definition at line 28 of file csv.h.

4.23.2.3 csvrecord_t

```
typedef struct csvrecord_s csvrecord_t
```

Definition at line 27 of file csv.h.

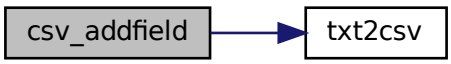
4.23.3 Function Documentation

4.23.3.1 csv_addfield()

```
void csv_addfield (
            FILE * reportfile,
            const char * value )
```

Definition at line 37 of file csv.c.

Here is the call graph for this function:



4.23.3.2 csv_addline()

```
void csv_addline (
            FILE * reportfile )
```

Definition at line 32 of file csv.c.

4.23.3.3 csvtok()

```
char * csvtok (
            char * source )
```

Definition at line 50 of file csv.c.

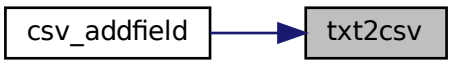
Here is the caller graph for this function:

4.23.3.4 txt2csv()

```
char * txt2csv (
            const char * text )
```

Definition at line 146 of file csv.c.

Here is the caller graph for this function:



4.24 csv.h

Go to the documentation of this file.

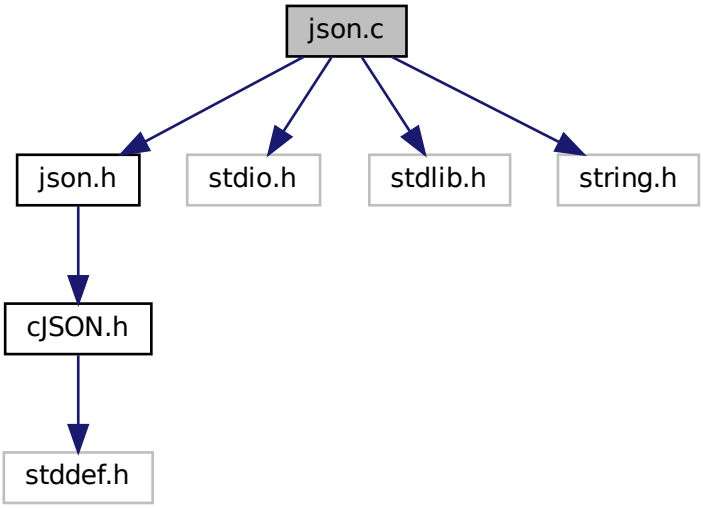
```
00001
00021 #ifndef __CSV_H__
00022 #define __CSV_H__
00023
00024 #include <stdio.h>                              /* FILE */
00025
00026 typedef struct csv_s csv_t;
00027 typedef struct csvrecord_s csvrecord_t;
00028 typedef struct csvfield_s csvfield_t;
00029
00030 void csv_addline(FILE* reportfile);
00031 void csv_addfield(FILE* reportfile, const char* value);
00032
00033 char* csvtok(char* source);
00034 char* txt2csv(const char* text);
00035
00036 #endif /* __CSV_H__ */
00037 /* vim:  set tw=80:  */
```

4.25 json.c File Reference

Wrapper around cJSON library with helpers.

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

Include dependency graph for json.c:



Functions

- char * json2text (cJSON *value_json)
  Convert a cJSON object in a C String.

4.25.1 Detailed Description

Wrapper around cJSON library with helpers.

This library is only a simple wrapper around the cJSON library, providing helper functions.

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file json.c.

4.25.2 Function Documentation

4.25.2.1 json2text()

```
char * json2text (
            cJSON * value_json )
```

Convert a cJSON object in a C String.

Parameters

value_json	the cJSON value to convert

Returns

The C string conversion dynamically allocated.

Return values

NULL	if there was a problem, such as an Out Of Memory
Pointer	to a zero terminated C string.

It converts NULL values, String values, Integer values, Boolean values in C strings. If the value is an array, it is fully pretty printed. If the JSON value is nothing in this list, the function returns the "Invalid data" string.

The returned strings are allocated using malloc on the heap and should be freed by the calling application.

Definition at line 31 of file json.c.

4.26 json.c

Go to the documentation of this file.

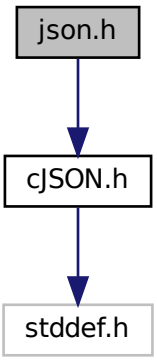
```
00001
00022 #ifdef HAVE_CONFIG_H
00023 #include "config.h"
00024 #endif
00025
00026 #include "json.h"
00027 #include <stdio.h>
00028 #include <stdlib.h>
00029 #include <string.h>
00030
00031 char* json2text(cJSON* value_json) {
00032     char* retval;
00033     if (cJSON_IsNull(value_json))
00034         retval = strdup("null");
00035     else if (cJSON_IsString(value_json))
00036         retval = strdup(value_json->valuestring);
00037     else if (cJSON_IsNumber(value_json)) {
00038         retval = (char*)malloc(20);
00039         if (retval)
00040             snprintf(retval,20,"%d",value_json->valueint);
00041     } else if (cJSON_IsBool(value_json)) {
00042         if (cJSON_IsTrue(value_json))
00043             retval = strdup("true");
```

```
00044         else
00045             retval = strdup("false");
00046     } else if (cJSON_IsArray(value_json)) {
00047         retval = cJSON_PrintUnformatted(value_json);
00048     } else if (cJSON_IsObject(value_json)) {
00049         retval = cJSON_PrintUnformatted(value_json);
00050     } else
00051         retval = strdup("Invalid data");
00052     return retval;
00053 }
00054
00055 /* vim:  set tw=80:  */
```

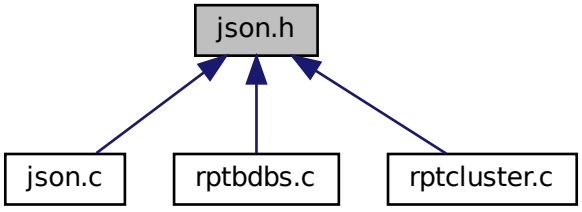
4.27 json.h File Reference

Wrapper around cJSON library with helpers.

`#include "cJSON.h"`
Include dependency graph for json.h:



This graph shows which files directly or indirectly include this file:



Functions

- char * json2text (cJSON *value_json)
  Convert a cJSON object in a C String.

4.27.1 Detailed Description

Wrapper around cJSON library with helpers.

This library is only a simple wrapper around the cJSON library, providing helper functions.

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file json.h.

4.27.2 Function Documentation

4.27.2.1 json2text()

```
char * json2text (
            cJSON * value_json )
```

Convert a cJSON object in a C String.

Parameters

value_json	the cJSON value to convert

Returns

The C string conversion dynamically allocated.

Return values

NULL	if there was a problem, such as an Out Of Memory
Pointer	to a zero terminated C string.

It converts NULL values, String values, Integer values, Boolean values in C strings. If the value is an array, it is fully pretty printed. If the JSON value is nothing in this list, the function returns the "Invalid data" string.

The returned strings are allocated using malloc on the heap and should be freed by the calling application.

Definition at line 31 of file json.c.

4.28 json.h

Go to the documentation of this file.

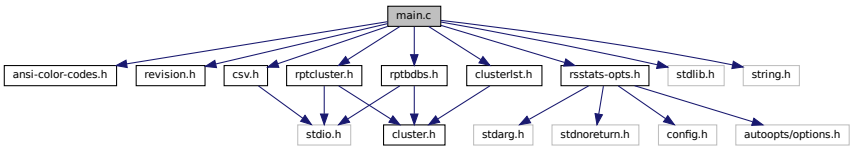
```
00001
00022 #ifndef __JSON_H__
00023 #define __JSON_H__
00024
00025 #include "cJSON.h"
00026
00043 char* json2text(cJSON* value_json);
00044
00045 #endif /* __JSON_H__ */
00046 /* vim:  set tw=80:  */
```

4.29 main.c File Reference

<+DETAILED+>

```
#include "ansi-color-codes.h"
#include "revision.h"
#include "csv.h"
#include "clusterlst.h"
#include "rsstats-opts.h"
#include "rptbdbs.h"
#include "rptcluster.h"
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for main.c:



Functions

- int main (int argc, char **argv, char **env)

4.29.1 Detailed Description

<+DETAILED+>

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file main.c.

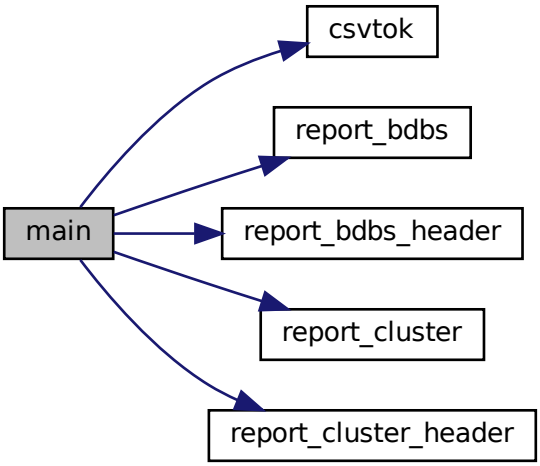
4.29.2 Function Documentation

4.29.2.1 main()

```
int main (
            int argc,
            char ** argv,
            char ** env )
```

Definition at line 38 of file main.c.

Here is the call graph for this function:



4.30 main.c

Go to the documentation of this file.

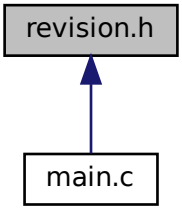
```
00001
00021 #ifdef HAVE_CONFIG_H
00022 #include "config.h"
00023 #endif
00024
00025 #include "ansi-color-codes.h"
00026 #include "revision.h"
00027 #include "csv.h"                          /* CSV manipulation functions */
00028 #include "clusterlst.h"                   /* Cluster list structure */
00029 #ifndef NOCPPCHECK
00030 #include "rsstats-opts.h"                 /* Libopts generated options */
00031 #endif
00032 #include "rptbdbs.h"
00033 #include "rptcluster.h"
00034
00035 #include <stdlib.h>
00036 #include <string.h>
00037
00038 int main (int argc, char** argv, char** env) {
```

```
00039     FILE *reportfile;
00040     FILE *configfile;
00041     char  configline[1024];
00042     cluster_t* cluster;
00043 
00044     (void)env;                                  /* Avoid unused warning/error */
00045     printf(BCYN PACKAGE_NAME " " PACKAGE_VERSION RESET "\n");
00046 #ifdef REVISION
00047     printf("Revision " BBLU REVISION RESET);
00048 #endif
00049 #ifdef BBID
00050     printf(" Build #" BBID);
00051 #endif
00052     printf("\n");
00053 
00054 #pragma GCC diagnostic push                         /* save the actual diag context */
00055 #pragma GCC diagnostic ignored "-Wdate-time"        /* locally disable warnings because of non
      reproductible build triggered by pbuild */
00056     printf("Compiled %s at %s\n",__DATE__, __TIME__);
00057 #pragma GCC diagnostic pop                          /* restore previous diag context */
00058     printf("Copyright 2024 François Cerbelle\n");
00059     printf("Report bugs to %s\n\n", BYEL PACKAGE_BUGREPORT RESET);
00060     /* AutoGen option parsing and consumming */
00061     {
00062         int arg_ct = optionProcess( &rsstatsOptions, argc, argv );
00063         argc -= arg_ct;
00064         argv += arg_ct;
00065     }
00066 
00067     printf("==> Read input file (cluster definitions):  %s\n",OPT_ARG(INPUT));
00068     /* Open the cluster definitions files */
00069     configfile = fopen(OPT_ARG(INPUT), "r");
00070     if (!configfile) {
00071         perror("main open(configfile)");
00072         exit(EXIT_FAILURE);
00073     }
00074 
00075     unsigned int lineno = 0;
00076     while (fgets(configline, sizeof(configline), configfile) != NULL) {
00077         lineno++;
00078 
00079         /* Allocate a cluster record */
00080         if(NULL==(cluster=malloc(sizeof(struct cluster_s)))) {
00081             perror("main malloc(cluster)");
00082             exit(EXIT_FAILURE);
00083         }
00084 
00085         /* Extract cluster information from the configline */
00086         if (NULL==(cluster->host = csvtok(configline))) {
00087             fprintf(stderr,"Bad configline (%s:%ud)\n", OPT_ARG(INPUT), lineno);
00088             free(cluster);
00089             continue;
00090         }
00091         if (NULL==(cluster->user = csvtok(NULL))) {
00092             fprintf(stderr,"Bad configline (%s:%ud)\n", OPT_ARG(INPUT), lineno);
00093             free(cluster->host);
00094             free(cluster);
00095             continue;
00096         }
00097         if (NULL==(cluster->pass = csvtok(NULL))) {
00098             fprintf(stderr,"Bad configline (%s:%ud)\n", OPT_ARG(INPUT), lineno);
00099             free(cluster->host);
00100             free(cluster->user);
00101             free(cluster);
00102             continue;
00103         }
00104         if (NULL==(cluster->insecure = csvtok(NULL))) {
00105             fprintf(stderr,"Bad configline (%s:%ud)\n", OPT_ARG(INPUT), lineno);
00106             free(cluster->host);
00107             free(cluster->user);
00108             free(cluster->pass);
00109             free(cluster);
00110             continue;
00111         }
00112         if (NULL==(cluster->cacert = csvtok(NULL))) {
00113             fprintf(stderr,"Bad configline (%s:%ud)\n", OPT_ARG(INPUT), lineno);
00114             free(cluster->host);
00115             free(cluster->user);
00116             free(cluster->pass);
00117             free(cluster->insecure);
00118             free(cluster);
00119             continue;
00120         }
00121 
00122         /* Check if the configline should be processed */
00123         if ((NULL!=strstr(OPT_ARG(CLUSTERS),"all"))
00124                 ||(NULL!=strstr(OPT_ARG(CLUSTERS),cluster->host)))
```

```
00125             cluster->enabled=1;
00126         else
00127             cluster->enabled=0;
00128         if (clusterlist_find(cluster->host))
00129             fprintf(stderr,"Double cluster definition (%s @ %s:%ud)\n",
00130                     cluster->host, OPT_ARG(INPUT), lineno);
00131         clusterlist_add(cluster);
00132     }
00133     fclose(configfile);
00134 
00135     printf("==> Clusters to query :  ");
00136     cluster=clusterlist_first();
00137     while(cluster) {
00138         if (cluster->enabled==1)
00139             printf("%s ",cluster->host);
00140         cluster = clusterlist_next();
00141     }
00142     printf("\n");
00143 
00144     printf("==> Open output file (report):  %s\n",OPT_ARG(OUTPUT));
00145 
00146     /* Open the output file */
00147     reportfile = fopen(OPT_ARG(OUTPUT),"w");
00148     if (!reportfile) {
00149         perror("Opening output file");
00150         exit(EXIT_FAILURE);
00151     }
00152 
00153     /* Execute cluster report ?  all or cluster specified */
00154     if (OPT_VALUE_REPORTS & REPORTS_CLUSTER) {
00155         printf("==> Running reports (cluster)\n");
00156         fprintf(reportfile,"\nclusters:\n");
00157         report_cluster_header(reportfile);
00158         /* Execute report against enabled clusters */
00159         cluster=clusterlist_first();
00160         while(cluster) {
00161             /* Check if the configline should be processed */
00162             if (cluster->enabled==1) {
00163                 printf("==> Running report (clusters) on cluster %s\n",cluster->host);
00164                 report_cluster(reportfile, cluster);
00165             }
00166             cluster = clusterlist_next();
00167         }
00168     }
00169 
00170     /* Execute bdbs report ?  all or bdbs specified */
00171     if (OPT_VALUE_REPORTS & REPORTS_BDBS) {
00172         printf("==> Running reports (bdbs)\n");
00173         fprintf(reportfile,"\nbdbs:\n");
00174         report_bdbs_header(reportfile);
00175         /* Execute report against enabled clusters */
00176         cluster=clusterlist_first();
00177         while(cluster) {
00178             /* Check if the configline should be processed */
00179             if (cluster->enabled==1) {
00180                 printf("==> Running report (bdbs) on cluster %s\n",cluster->host);
00181                 report_bdbs(reportfile, cluster);
00182             }
00183             cluster = clusterlist_next();
00184         }
00185     }
00186 
00187     fclose(reportfile);
00188 
00189 #ifdef _WIN32
00190     system("PAUSE");  /* For windows console window to wait.  */
00191 #endif
00192 
00193     return EXIT_SUCCESS;
00194 }
00195 /* vim:  set tw=80:  */
```

4.31 revision.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define REVISION "1f1d73a22c22"

4.31.1 Macro Definition Documentation

4.31.1.1 REVISION

```
#define REVISION "1f1d73a22c22"
```

Definition at line 3 of file revision.h.

4.32 revision.h

Go to the documentation of this file.

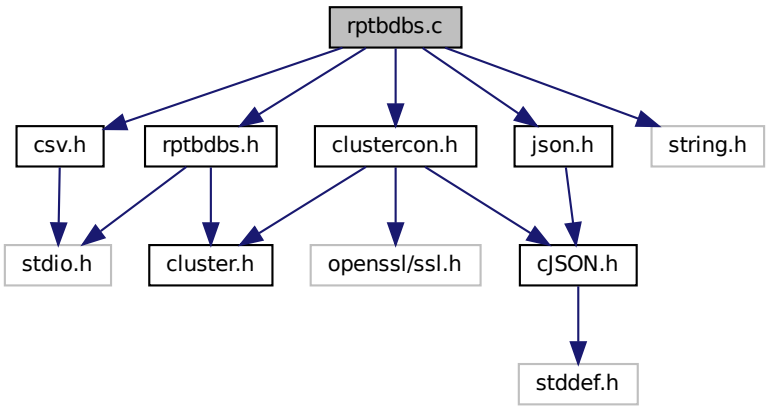
```
00001 /* This file is updated in the distdir before creating the dist archive */
00002 #ifndef REVISION
00003 #define REVISION "1f1d73a22c22"
00004 #endif
```

4.33 rptbdbs.c File Reference

<+DETAILED+>

```
#include "rptbdbs.h"
#include "clustercon.h"
#include "json.h"
#include "csv.h"
#include <string.h>
```

Include dependency graph for rptbdbs.c:



Functions

- void report_bdbs_header (FILE *reportfile)
- void report_bdbs (FILE *reportfile, const cluster_t *cluster)

4.33.1 Detailed Description

<+DETAILED+>

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file rptbdbs.c.

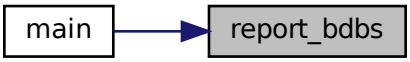
4.33.2 Function Documentation

4.33.2.1 report_bdbs()

```
void report_bdbs (
            FILE * reportfile,
            const cluster_t * cluster )
```

Definition at line 71 of file rptbdbs.c.

Here is the caller graph for this function:



4.33.2.2 report_bdbs_header()

```
void report_bdbs_header (
            FILE * reportfile )
```

Definition at line 38 of file rptbdbs.c.

Here is the caller graph for this function:

4.34 rptbdbs.c

Go to the documentation of this file.

```
00001
00021 #ifdef HAVE_CONFIG_H
00022 #include "config.h"
00023 #endif
00024
00025 #include "rptbdbs.h"
00026 #include "clustercon.h"
00027 #include "json.h"
00028 #include "csv.h"
00029 #include <string.h>
00030
00031 /* To refactor in a report common file */
```

```
00032 static void csv_addjsonfield(FILE* reportfile, const cJSON* json, char* fieldname) {
00033     char* text = json2text(cJSON_GetObjectItemCaseSensitive(json, fieldname));
00034     csv_addfield(reportfile,text);
00035     free(text);
00036 }
00037
00038 void report_bdbs_header(FILE* reportfile) {
00039     fprintf(reportfile,
00040
      "cluster_host,uid,name,shards_count,replication,data_persistence,memory_size,used_memory,module_list\r\n"
00041           );
00042 }
00043
00044 static cJSON* report_querygetjson(const cluster_t* cluster, const char* endpoint) {
00045     rsclustercon_t* rsclustercon;
00046
00047     if (NULL==(rsclustercon = cluster_new(cluster))) {
00048         fprintf(stderr,"report_querygetjson cluster_new failed\n");
00049         return NULL;
00050     }
00051     if (0!=cluster_open(rsclustercon)) {
00052         fprintf(stderr,"report_querygetjson cluster_open failed\n");
00053         cluster_del(rsclustercon);
00054         return NULL;
00055     }
00056
00057     cJSON* retval;
00058     if (NULL==(retval = cluster_queryget(rsclustercon, endpoint))) {
00059         const char *error_ptr = cJSON_GetErrorPtr();
00060         if (error_ptr != NULL)
00061             fprintf(stderr, "Error before:  %s\n", error_ptr);
00062         retval = cJSON_CreateObject();
00063     }
00064
00065     cluster_close(rsclustercon);
00066     cluster_del(rsclustercon);
00067
00068     return retval;
00069 }
00070
00071 void report_bdbs(FILE* reportfile, const cluster_t* cluster) {
00072     cJSON* bdbs_json;
00073     cJSON* bdbsstats_json;
00074
00075     bdbs_json = report_querygetjson(cluster, "/v1/bdbs");
00076     bdbsstats_json = report_querygetjson(cluster, "/v1/bdbs/stats/last");
00077
00078     const cJSON* bdb_json;
00079     cJSON_ArrayForEach(bdb_json, bdbs_json) {
00080         char* uid=json2text(cJSON_GetObjectItemCaseSensitive(bdb_json, "uid"));
00081         cJSON* stats_json = cJSON_GetObjectItemCaseSensitive(bdbsstats_json, uid);
00082         free(uid);
00083         csv_addfield(reportfile,cluster->host);
00084         csv_addjsonfield(reportfile, bdb_json, "uid");
00085         csv_addjsonfield(reportfile, bdb_json, "name");
00086         csv_addjsonfield(reportfile, bdb_json, "shards_count");
00087         csv_addjsonfield(reportfile, bdb_json, "replication");
00088         csv_addjsonfield(reportfile, bdb_json, "data_persistence");
00089         csv_addjsonfield(reportfile, bdb_json, "memory_size");
00090         csv_addjsonfield(reportfile, stats_json, "used_memory");
00091
00092         /* Iterate the module list and build the text list */
00093         char* modlst;
00094         if (NULL==(modlst=strdup(""))) {
00095             perror("rptbdbs repport_bdbs module_list");
00096             exit(EXIT_FAILURE);
00097         }
00098
00099         const cJSON* module_json;
00100         cJSON_ArrayForEach(module_json, cJSON_GetObjectItemCaseSensitive(bdb_json, "module_list")) {
00101             char* modulename = cJSON_GetObjectItemCaseSensitive(module_json,
      "module_name")->valuestring;
00102             char* newmodlst;
00103             if (NULL==(newmodlst =
      (char*)realloc(modlst,strlen(modlst)+strlen(modulename?modulename:"")+2+1))) {
00104                 perror("rptbdbs repport_bdbs module_list");
00105                 free(modlst);
00106                 return;
00107             } else
00108                 modlst = newmodlst;
00109             strcat(modlst,(modulename?modulename:""));
00110             strcat(modlst,", ");
00111         }
00112
00113         /* Remove last comma if applicable */
00114         if (strlen(modlst)>2)
00115             modlst[strlen(modlst)-2]=0;
```

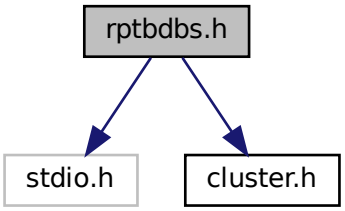
```
00116         csv_addfield(reportfile,modlst);
00117         free(modlst);
00118
00119         csv_addline(reportfile);
00120     }
00121     cJSON_Delete(bdbs_json);
00122     cJSON_Delete(bdbsstats_json);
00123 }
00124 /* vim:  set tw=80:  */
```

4.35 rptbdbs.h File Reference

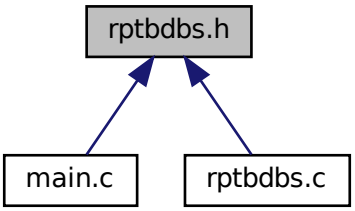
<+DETAILED+>

```
#include <stdio.h>
#include "cluster.h"
```

Include dependency graph for rptbdbs.h:



This graph shows which files directly or indirectly include this file:



Functions

- void report_bdbs (FILE *reportfile, const cluster_t *cluster)
- void report_bdbs_header (FILE *reportfile)

4.35.1 Detailed Description

<+DETAILED+>

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file rptbdbs.h.

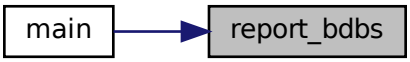
4.35.2 Function Documentation

4.35.2.1 report_bdbs()

```
void report_bdbs (
            FILE * reportfile,
            const cluster_t * cluster )
```

Definition at line 71 of file rptbdbs.c.

Here is the caller graph for this function:



4.35.2.2 report_bdbs_header()

```
void report_bdbs_header (
            FILE * reportfile )
```

Definition at line 38 of file rptbdbs.c.

Here is the caller graph for this function:

4.36 rptbdbs.h

Go to the documentation of this file.

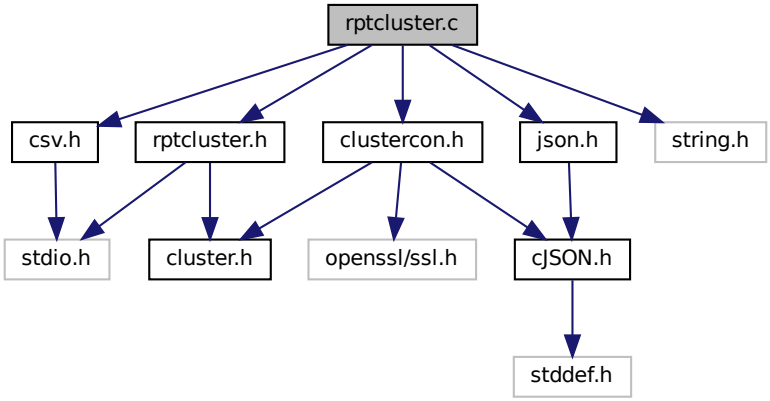
```
00001
00021 #ifndef __RPTBDBS_H__
00022 #define __RPTBDBS_H__
00023
00024 #include <stdio.h>
00025 #include "cluster.h"
00026
00027 void report_bdbs(FILE* reportfile, const cluster_t* cluster);
00028 void report_bdbs_header(FILE* reportfile);
00029
00030 #endif /* __RPTBDBS_H__ */
00031 /* vim:  set tw=80:  */
```

4.37 rptcluster.c File Reference

<+DETAILED+>

```
#include "rptcluster.h"
#include "clustercon.h"
#include "json.h"
#include "csv.h"
#include <string.h>
```

Include dependency graph for rptcluster.c:



Functions

- void report_cluster_header (FILE *reportfile)
- void report_cluster (FILE *reportfile, const cluster_t *cluster)

4.37.1 Detailed Description

<+DETAILED+>

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file rptcluster.c.

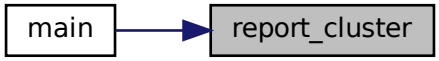
4.37.2 Function Documentation

4.37.2.1 report_cluster()

```
void report_cluster (
            FILE * reportfile,
            const cluster_t * cluster )
```

Definition at line 79 of file rptcluster.c.

Here is the caller graph for this function:



4.37.2.2 report_cluster_header()

```
void report_cluster_header (
            FILE * reportfile )
```

Definition at line 38 of file rptcluster.c.

Here is the caller graph for this function:

4.38 rptcluster.c

Go to the documentation of this file.

```
00001
00021 #ifdef HAVE_CONFIG_H
00022 #include "config.h"
00023 #endif
00024
00025 #include "rptcluster.h"
00026 #include "clustercon.h"
00027 #include "json.h"
00028 #include "csv.h"
00029 #include <string.h>
00030
00031 /* To refactor in a report common file */
00032 static void csv_addjsonfield(FILE* reportfile, const cJSON* json, char* fieldname) {
00033     char* text = json2text(cJSON_GetObjectItemCaseSensitive(json, fieldname));
00034     csv_addfield(reportfile,text);
00035     free(text);
00036 }
00037
00038 void report_cluster_header(FILE* reportfile) {
00039     fprintf(reportfile,
00040             "cluster_host,name,rack_aware,created_time,free_memory,"
00041             "available_memory,available_memory_no_overbooking,available_flash,"
00042             "available_flash_no_overbooking,ephemeral_storage_avail,"
00043             "ephemeral_storage_free,persistent_storage_avail,"
00044             "persistent_storage_free,provisional_flash,"
00045             "provisional_flash_no_overbooking,provisional_memory,"
00046             "provisional_memory_no_overbooking,activation_date,expiration_date,"
00047             "cluster_name,owner,features,shards_limit,ram_shards_in_use,"
00048             "ram_shards_limit,flash_shards_in_use,flash_shards_limit,expired\r\n"
00049            );
00050 }
00051
00052 static cJSON* report_querygetjson(const cluster_t* cluster, const char* endpoint) {
00053     rsclustercon_t* rsclustercon;
00054
00055     if (NULL==(rsclustercon = cluster_new(cluster))) {
00056         fprintf(stderr,"report_querygetjson cluster_new failed\n");
00057         return NULL;
00058     }
00059     if (0!=cluster_open(rsclustercon)) {
00060         fprintf(stderr,"report_querygetjson cluster_open failed\n");
00061         cluster_del(rsclustercon);
00062         return NULL;
00063     }
00064
00065     cJSON* retval;
00066     if (NULL==(retval = cluster_queryget(rsclustercon, endpoint))) {
00067         const char *error_ptr = cJSON_GetErrorPtr();
00068         if (error_ptr != NULL)
00069             fprintf(stderr, "Error before:  %s\n", error_ptr);
00070         retval = cJSON_CreateObject();
00071     }
00072
00073     cluster_close(rsclustercon);
00074     cluster_del(rsclustercon);
00075
00076     return retval;
00077 }
00078
00079 void report_cluster(FILE* reportfile, const cluster_t* cluster) {
00080     cJSON* cluster_json;
00081     cJSON* clusterstats_json;
00082     cJSON* license_json;
00083
00084     cluster_json = report_querygetjson(cluster, "/v1/cluster");
00085     clusterstats_json = report_querygetjson(cluster, "/v1/cluster/stats/last");
00086     license_json = report_querygetjson(cluster, "/v1/license");
00087
00088     csv_addfield(reportfile,cluster->host);
00089     csv_addjsonfield(reportfile, cluster_json, "name");
00090     csv_addjsonfield(reportfile, cluster_json, "rack_aware");
00091     csv_addjsonfield(reportfile, cluster_json, "created_time");
00092     csv_addjsonfield(reportfile, clusterstats_json, "free_memory");
00093     csv_addjsonfield(reportfile, clusterstats_json, "available_memory");
00094     csv_addjsonfield(reportfile, clusterstats_json, "available_memory_no_overbooking");
00095     csv_addjsonfield(reportfile, clusterstats_json, "available_flash");
00096     csv_addjsonfield(reportfile, clusterstats_json, "available_flash_no_overbooking");
00097     csv_addjsonfield(reportfile, clusterstats_json, "ephemeral_storage_avail");
00098     csv_addjsonfield(reportfile, clusterstats_json, "ephemeral_storage_free");
00099     csv_addjsonfield(reportfile, clusterstats_json, "persistent_storage_avail");
00100     csv_addjsonfield(reportfile, clusterstats_json, "persistent_storage_free");
00101     csv_addjsonfield(reportfile, clusterstats_json, "provisional_flash");
```

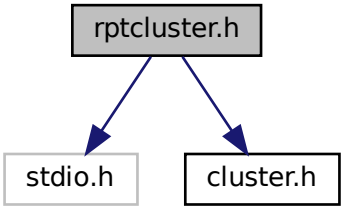
```
00102     csv_addjsonfield(reportfile, clusterstats_json, "provisional_flash_no_overbooking");
00103     csv_addjsonfield(reportfile, clusterstats_json, "provisional_memory");
00104     csv_addjsonfield(reportfile, clusterstats_json, "provisional_memory_no_overbooking");
00105     csv_addjsonfield(reportfile, license_json, "activation_date");
00106     csv_addjsonfield(reportfile, license_json, "expiration_date");
00107     csv_addjsonfield(reportfile, license_json, "cluster_name");
00108     csv_addjsonfield(reportfile, license_json, "owner");
00109     csv_addjsonfield(reportfile, license_json, "features");
00110     csv_addjsonfield(reportfile, license_json, "shards_limit");
00111     csv_addjsonfield(reportfile, license_json, "ram_shards_in_use");
00112     csv_addjsonfield(reportfile, license_json, "ram_shards_limit");
00113     csv_addjsonfield(reportfile, license_json, "flash_shards_in_use");
00114     csv_addjsonfield(reportfile, license_json, "flash_shards_limit");
00115     csv_addjsonfield(reportfile, license_json, "expired");
00116     csv_addline(reportfile);
00117
00118     cJSON_Delete(cluster_json);
00119     cJSON_Delete(clusterstats_json);
00120     cJSON_Delete(license_json);
00121 }
00122
00123
00124 /*
00125
00126
00127
00128 clusterdef.host
00129 nodes.uid
00130 nodes.addr
00131 nodes.external_addr
00132 nodes.cores
00133 nodes.total_memory
00134 nodes.ephemeral_storage_size
00135 nodes.persistent_storage_size
00136 nodes.os_version
00137 nodes.rack_id
00138 nodes.max_listeners
00139 nodes.max_redis_servers
00140 nodes.ram_shard_count
00141 nodes.flash_shard_count
00142 nodes.shard_count
00143 nodes.software_version
00144 nodes.uptime
00145 nodes.accept_servers
00146 nodes.status
00147 nodesstatus.cores
00148 nodesstatus.free_provisional_ram
00149 nodesstatus.free_ram
00150 nodesstatus.hostname
00151 nodesstatus.node_overbooking_depth
00152 nodesstatus.node_status
00153 nodesstatus.role
00154 nodesstatus.software_version
00155 nodesstatus.total_memory
00156 nodesstatus.total_provisional_ram
00157 nodesstatslast.available_memory
00158 nodesstatslast.available_memory_no_overbooking
00159 nodesstatslast.ephemeral_storage_avail
00160 nodesstatslast.ephemeral_storage_free
00161 nodesstatslast.free_memory
00162 nodesstatslast.persistent_storage_avail
00163 nodesstatslast.persistent_storage_free
00164 nodesstatslast.provisional_memory
00165 nodesstatslast.provisional_memory_no_overbooking
00166 */
00167 /* vim:  set tw=80:  */
```

4.39 rptcluster.h File Reference

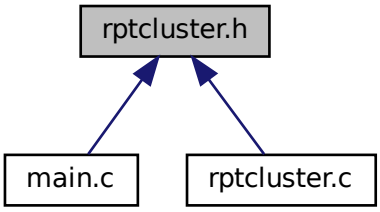
<+DETAILED+>

```
#include <stdio.h>
#include "cluster.h"
```

Include dependency graph for rptcluster.h:



This graph shows which files directly or indirectly include this file:



Functions

- void report_cluster (FILE *reportfile, const cluster_t *cluster)
- void report_cluster_header (FILE *reportfile)

4.39.1 Detailed Description

<+DETAILED+>

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file rptcluster.h.

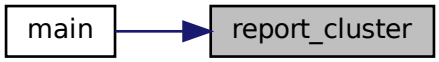
4.39.2 Function Documentation

4.39.2.1 report_cluster()

```
void report_cluster (
            FILE * reportfile,
            const cluster_t * cluster )
```

Definition at line 79 of file rptcluster.c.

Here is the caller graph for this function:



4.39.2.2 report_cluster_header()

```
void report_cluster_header (
            FILE * reportfile )
```

Definition at line 38 of file rptcluster.c.

Here is the caller graph for this function:

4.40 rptcluster.h

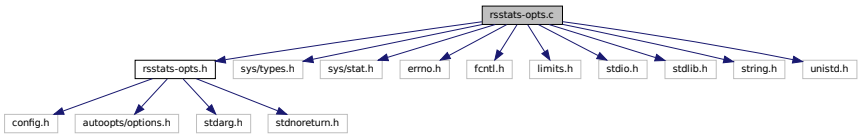
Go to the documentation of this file.

```
00001
00021 #ifndef __RPTCLUSTER_H__
00022 #define __RPTCLUSTER_H__
00023
00024 #include <stdio.h>
00025 #include "cluster.h"
00026
00027 void report_cluster(FILE* reportfile, const cluster_t* cluster);
00028 void report_cluster_header(FILE* reportfile);
00029
00030 #endif /* __RPTCLUSTER_H__ */
00031 /* vim:  set tw=80:  */
```

4.41 rsstats-opts.c File Reference

```
#include "rsstats-opts.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>
#include <fcntl.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

Include dependency graph for rsstats-opts.c:



Macros

- #define OPTION_CODE_COMPILE 1
- #define zCopyright (rsstats_opt_strs+0)
- #define zLicenseDescrip (rsstats_opt_strs+258)
- #define NULL 0
- #define INPUT_DESC (rsstats_opt_strs+861)

  input option description:
- #define INPUT_NAME (rsstats_opt_strs+902)

  Upper-cased name for the input option.
- #define INPUT_name (rsstats_opt_strs+908)

  Name string for the input option.
- #define INPUT_DFT_ARG (rsstats_opt_strs+914)

  The compiled in default value for the input option argument.
- #define INPUT_FLAGS

  Compiled in flag settings for the input option.
- #define OUTPUT_DESC (rsstats_opt_strs+929)

  output option description:
- #define OUTPUT_NAME (rsstats_opt_strs+994)

  Upper-cased name for the output option.
- #define OUTPUT_name (rsstats_opt_strs+1001)

  Name string for the output option.
- #define OUTPUT_DFT_ARG (rsstats_opt_strs+1008)

  The compiled in default value for the output option argument.
- #define OUTPUT_FLAGS

  Compiled in flag settings for the output option.
- #define CLUSTERS_DESC (rsstats_opt_strs+1020)

  clusters option description:
- #define CLUSTERS_NAME (rsstats_opt_strs+1081)

  Upper-cased name for the clusters option.

- #define CLUSTERS_name (rsstats_opt_strs+1090)

  Name string for the clusters option.
- #define CLUSTERS_DFT_ARG (rsstats_opt_strs+1099)

  The compiled in default value for the clusters option argument.
- #define CLUSTERS_FLAGS

  Compiled in flag settings for the clusters option.
- #define REPORTS_DESC (rsstats_opt_strs+1103)

  reports option description:
- #define REPORTS_NAME (rsstats_opt_strs+1162)

  Upper-cased name for the reports option.
- #define REPORTS_name (rsstats_opt_strs+1170)

  Name string for the reports option.
- #define REPORTS_DFT_ARG (NULL)

  The compiled in default value for the reports option argument.
- #define ReportsCookieBits VOIDP(REPORTS_BDBS|REPORTS_CLUSTER)
- #define REPORTS_FLAGS

  Compiled in flag settings for the reports option.
- #define HELP_DESC (rsstats_opt_strs+1178)
- #define HELP_name (rsstats_opt_strs+1222)
- #define MORE_HELP_DESC HELP_DESC
- #define MORE_HELP_name HELP_name
- #define MORE_HELP_FLAGS (OPTST_OMITTED | OPTST_NO_INIT)
- #define VER_FLAGS
- #define VER_DESC (rsstats_opt_strs+1282)
- #define VER_name (rsstats_opt_strs+1318)
- #define SAVE_OPTS_DESC (rsstats_opt_strs+1326)
- #define SAVE_OPTS_name (rsstats_opt_strs+1365)
- #define LOAD_OPTS_DESC (rsstats_opt_strs+1375)
- #define LOAD_OPTS_NAME (rsstats_opt_strs+1407)
- #define NO_LOAD_OPTS_name (rsstats_opt_strs+1417)
- #define LOAD_OPTS_pfx (rsstats_opt_strs+1430)
- #define LOAD_OPTS_name (NO_LOAD_OPTS_name + 3)
- #define VER_PROC optionPrintVersion
- #define zPROGNAME (rsstats_opt_strs+1433)

  Reference to the upper cased version of rsstats.
- #define zUsageTitle (rsstats_opt_strs+1441)

  Reference to the title line for rsstats usage.
- #define zRcName (rsstats_opt_strs+1570)

  rsstats configuration file name.
- #define zBugsAddr (rsstats_opt_strs+1581)

  The rsstats program bug email address.
- #define zExplain (rsstats_opt_strs+1603)

  Clarification/explanation of what rsstats does.
- #define zDetail (rsstats_opt_strs+1672)

  Extra detail explaining what rsstats does.
- #define zFullVersion (rsstats_opt_strs+1859)

  The full version string for rsstats.
- #define OPTPROC_BASE OPTPROC_NONE
- #define translate_option_strings NULL
- #define rsstats_full_usage (NULL)
- #define rsstats_short_usage (NULL)
- #define O_CLOEXEC 0

- #define PKGDATADIR ""

  The directory containing the data associated with rsstats.
- #define rsstats_packager_info NULL

  Information about the person or institution that packaged rsstats for the current distribution.

Variables

- FILE * option_usage_fp
- tOptProc optionBooleanVal

  Declare option callback procedures.
- tOptProc optionNestedVal
- tOptProc optionNumericVal
- tOptProc optionPagedUsage
- tOptProc optionPrintVersion
- tOptProc optionResetOpt
- tOptProc optionStackArg
- tOptProc optionTimeDate
- tOptProc optionTimeVal
- tOptProc optionUnstackArg
- tOptProc optionVendorOption
- tOptions rsstatsOptions

  The option definitions for rsstats.

4.41.1 Macro Definition Documentation

4.41.1.1 CLUSTERS_DESC

```
#define CLUSTERS_DESC (rsstats_opt_strs+1020)
```

clusters option description:

Descriptive text for the clusters option

Definition at line 159 of file rsstats-opts.c.

4.41.1.2 CLUSTERS_DFT_ARG

```
#define CLUSTERS_DFT_ARG (rsstats_opt_strs+1099)
```

The compiled in default value for the clusters option argument.

Definition at line 165 of file rsstats-opts.c.

4.41.1.3 CLUSTERS_FLAGS

```
#define CLUSTERS_FLAGS
```

Value:

```
(OPTST_DISABLED \
| OPTST_SET_ARGTYPE(OPARG_TYPE_STRING))
```

Compiled in flag settings for the clusters option.

Definition at line 167 of file rsstats-opts.c.

4.41.1.4 CLUSTERS_NAME

```
#define CLUSTERS_NAME (rsstats_opt_strs+1081)
```

Upper-cased name for the clusters option.

Definition at line 161 of file rsstats-opts.c.

4.41.1.5 CLUSTERS_name

```
#define CLUSTERS_name (rsstats_opt_strs+1090)
```

Name string for the clusters option.

Definition at line 163 of file rsstats-opts.c.

4.41.1.6 HELP_DESC

```
#define HELP_DESC (rsstats_opt_strs+1178)
```

Definition at line 189 of file rsstats-opts.c.

4.41.1.7 HELP_name

```
#define HELP_name (rsstats_opt_strs+1222)
```

Definition at line 190 of file rsstats-opts.c.

4.41.1.8 INPUT_DESC

```
#define INPUT_DESC (rsstats_opt_strs+861)
```

input option description:

Descriptive text for the input option

Definition at line 129 of file rsstats-opts.c.

4.41.1.9 INPUT_DFT_ARG

```
#define INPUT_DFT_ARG (rsstats_opt_strs+914)
```

The compiled in default value for the input option argument.

Definition at line 135 of file rsstats-opts.c.

4.41.1.10 INPUT_FLAGS

```
#define INPUT_FLAGS
```

Value:

```
        (OPTST_DISABLED \
        | OPTST_SET_ARGTYPE(OPARG_TYPE_FILE))
```

Compiled in flag settings for the input option.

Definition at line 137 of file rsstats-opts.c.

4.41.1.11 INPUT_NAME

```
#define INPUT_NAME (rsstats_opt_strs+902)
```

Upper-cased name for the input option.

Definition at line 131 of file rsstats-opts.c.

4.41.1.12 INPUT_name

```
#define INPUT_name (rsstats_opt_strs+908)
```

Name string for the input option.

Definition at line 133 of file rsstats-opts.c.

4.41.1.13 LOAD_OPTS_DESC

```
#define LOAD_OPTS_DESC (rsstats_opt_strs+1375)
```

Definition at line 210 of file rsstats-opts.c.

4.41.1.14 LOAD_OPTS_NAME

```
#define LOAD_OPTS_NAME (rsstats_opt_strs+1407)
```

Definition at line 211 of file rsstats-opts.c.

4.41.1.15 LOAD_OPTS_name

```
#define LOAD_OPTS_name (NO_LOAD_OPTS_name + 3)
```

Definition at line 214 of file rsstats-opts.c.

4.41.1.16 LOAD_OPTS_pfx

```
#define LOAD_OPTS_pfx (rsstats_opt_strs+1430)
```

Definition at line 213 of file rsstats-opts.c.

4.41.1.17 MORE_HELP_DESC

```
#define MORE_HELP_DESC HELP_DESC
```

Definition at line 196 of file rsstats-opts.c.

4.41.1.18 MORE_HELP_FLAGS

```
#define MORE_HELP_FLAGS (OPTST_OMITTED | OPTST_NO_INIT)
```

Definition at line 198 of file rsstats-opts.c.

4.41.1.19 MORE_HELP_name

```
#define MORE_HELP_name HELP_name
```

Definition at line 197 of file rsstats-opts.c.

4.41.1.20 NO_LOAD_OPTS_name

```
#define NO_LOAD_OPTS_name (rsstats_opt_strs+1417)
```

Definition at line 212 of file rsstats-opts.c.

4.41.1.21 NULL

```
#define NULL 0
```

Definition at line 64 of file rsstats-opts.c.

4.41.1.22 O_CLOEXEC

```
#define O_CLOEXEC 0
```

4.41.1.23 OPTION_CODE_COMPILE

```
#define OPTION_CODE_COMPILE 1
```

Definition at line 42 of file rsstats-opts.c.

4.41.1.24 OPTPROC_BASE

```
#define OPTPROC_BASE OPTPROC_NONE
```

Definition at line 373 of file rsstats-opts.c.

4.41.1.25 OUTPUT_DESC

```
#define OUTPUT_DESC (rsstats_opt_strs+929)
```

output option description:

Descriptive text for the output option

Definition at line 144 of file rsstats-opts.c.

4.41.1.26 OUTPUT_DFT_ARG

```
#define OUTPUT_DFT_ARG (rsstats_opt_strs+1008)
```

The compiled in default value for the output option argument.

Definition at line 150 of file rsstats-opts.c.

4.41.1.27 OUTPUT_FLAGS

```
#define OUTPUT_FLAGS
```

Value:

```
(OPTST_DISABLED \
| OPTST_SET_ARGTYPE(OPARG_TYPE_FILE))
```

Compiled in flag settings for the output option.

Definition at line 152 of file rsstats-opts.c.

4.41.1.28 OUTPUT_NAME

```
#define OUTPUT_NAME (rsstats_opt_strs+994)
```

Upper-cased name for the output option.

Definition at line 146 of file rsstats-opts.c.

4.41.1.29 OUTPUT_name

```
#define OUTPUT_name (rsstats_opt_strs+1001)
```

Name string for the output option.

Definition at line 148 of file rsstats-opts.c.

4.41.1.30 PKGDATADIR

```
#define PKGDATADIR ""
```

The directory containing the data associated with rsstats.

Definition at line 478 of file rsstats-opts.c.

4.41.1.31 REPORTS_DESC

```
#define REPORTS_DESC (rsstats_opt_strs+1103)
```

reports option description:

Descriptive text for the reports option

Definition at line 174 of file rsstats-opts.c.

4.41.1.32 REPORTS_DFT_ARG

```
#define REPORTS_DFT_ARG (NULL)
```

The compiled in default value for the reports option argument.

Definition at line 180 of file rsstats-opts.c.

4.41.1.33 REPORTS_FLAGS

```
#define REPORTS_FLAGS
```

Value:

```
        (OPTST_DISABLED \
        | OPTST_SET_ARGTYPE(OPARG_TYPE_MEMBERSHIP))
```

Compiled in flag settings for the reports option.

Definition at line 183 of file rsstats-opts.c.

4.41.1.34 REPORTS_NAME

```
#define REPORTS_NAME (rsstats_opt_strs+1162)
```

Upper-cased name for the reports option.

Definition at line 176 of file rsstats-opts.c.

4.41.1.35 REPORTS_name

```
#define REPORTS_name (rsstats_opt_strs+1170)
```

Name string for the reports option.

Definition at line 178 of file rsstats-opts.c.

4.41.1.36 ReportsCookieBits

```
#define ReportsCookieBits VOIDP(REPORTS_BDBS|REPORTS_CLUSTER)
```

Definition at line 181 of file rsstats-opts.c.

4.41.1.37 rsstats_full_usage

```
#define rsstats_full_usage (NULL)
```

Definition at line 377 of file rsstats-opts.c.

4.41.1.38 rsstats_packager_info

```
#define rsstats_packager_info NULL
```

Information about the person or institution that packaged rsstats for the current distribution.

Definition at line 486 of file rsstats-opts.c.

4.41.1.39 rsstats_short_usage

```
#define rsstats_short_usage (NULL)
```

Definition at line 378 of file rsstats-opts.c.

4.41.1.40 SAVE_OPTS_DESC

```
#define SAVE_OPTS_DESC (rsstats_opt_strs+1326)
```

Definition at line 208 of file rsstats-opts.c.

4.41.1.41 SAVE_OPTS_name

```
#define SAVE_OPTS_name (rsstats_opt_strs+1365)
```

Definition at line 209 of file rsstats-opts.c.

4.41.1.42 translate_option_strings

```
#define translate_option_strings NULL
```

Definition at line 374 of file rsstats-opts.c.

4.41.1.43 VER_DESC

```
#define VER_DESC (rsstats_opt_strs+1282)
```

Definition at line 206 of file rsstats-opts.c.

4.41.1.44 VER_FLAGS

```
#define VER_FLAGS
```

Value:

```
(OPTST_SET_ARGTYPE(OPARG_TYPE_STRING) | \
OPTST_ARG_OPTIONAL | OPTST_IMM | OPTST_NO_INIT)
```

Definition at line 203 of file rsstats-opts.c.

4.41.1.45 VER_name

```
#define VER_name (rsstats_opt_strs+1318)
```

Definition at line 207 of file rsstats-opts.c.

4.41.1.46 VER_PROC

```
#define VER_PROC optionPrintVersion
```

Definition at line 225 of file rsstats-opts.c.

4.41.1.47 zBugsAddr

```
#define zBugsAddr (rsstats_opt_strs+1581)
```

The rsstats program bug email address.

Definition at line 360 of file rsstats-opts.c.

4.41.1.48 zCopyright

```
#define zCopyright (rsstats_opt_strs+0)
```

Definition at line 59 of file rsstats-opts.c.

4.41.1.49 zDetail

```
#define zDetail (rsstats_opt_strs+1672)
```

Extra detail explaining what rsstats does.

Definition at line 364 of file rsstats-opts.c.

4.41.1.50 zExplain

```
#define zExplain (rsstats_opt_strs+1603)
```

Clarification/explanation of what rsstats does.

Definition at line 362 of file rsstats-opts.c.

4.41.1.51 zFullVersion

```
#define zFullVersion (rsstats_opt_strs+1859)
```

The full version string for rsstats.

Definition at line 366 of file rsstats-opts.c.

4.41.1.52 zLicenseDescrip

```
#define zLicenseDescrip (rsstats_opt_strs+258)
```

Definition at line 60 of file rsstats-opts.c.

4.41.1.53 zPROGNAME

```
#define zPROGNAME (rsstats_opt_strs+1433)
```

Reference to the upper cased version of rsstats.

Definition at line 350 of file rsstats-opts.c.

4.41.1.54 zRcName

```
#define zRcName (rsstats_opt_strs+1570)
```

rsstats configuration file name.

Definition at line 354 of file rsstats-opts.c.

4.41.1.55 zUsageTitle

```
#define zUsageTitle (rsstats_opt_strs+1441)
```

Reference to the title line for rsstats usage.

Definition at line 352 of file rsstats-opts.c.

4.41.2 Variable Documentation

4.41.2.1 option_usage_fp

```
FILE* option_usage_fp  [extern]
```

4.41.2.2 optionBooleanVal

```
tOptProc optionBooleanVal  [extern]
```

Declare option callback procedures.

4.41.2.3 optionNestedVal

```
tOptProc optionNestedVal
```

Definition at line 219 of file rsstats-opts.c.

4.41.2.4 optionNumericVal

```
tOptProc optionNumericVal
```

Definition at line 219 of file rsstats-opts.c.

4.41.2.5 optionPagedUsage

```
tOptProc optionPagedUsage
```

Definition at line 220 of file rsstats-opts.c.

4.41.2.6 optionPrintVersion

```
tOptProc optionPrintVersion
```

Definition at line 220 of file rsstats-opts.c.

4.41.2.7 optionResetOpt

```
tOptProc optionResetOpt
```

Definition at line 220 of file rsstats-opts.c.

4.41.2.8 optionStackArg

```
tOptProc optionStackArg
```

Definition at line 221 of file rsstats-opts.c.

4.41.2.9 optionTimeDate

```
tOptProc optionTimeDate
```

Definition at line 221 of file rsstats-opts.c.

4.41.2.10 optionTimeVal

```
tOptProc optionTimeVal
```

Definition at line 221 of file rsstats-opts.c.

4.41.2.11 optionUnstackArg

```
tOptProc optionUnstackArg
```

Definition at line 222 of file rsstats-opts.c.

4.41.2.12 optionVendorOption

```
tOptProc optionVendorOption
```

Definition at line 222 of file rsstats-opts.c.

4.41.2.13 rsstatsOptions

```
tOptions rsstatsOptions
```

The option definitions for rsstats.

The one structure that binds them all.

Definition at line 508 of file rsstats-opts.c.

4.42 rsstats-opts.c

Go to the documentation of this file.

```
00001 /*   -*- buffer-read-only:  t -*- vi:  set ro:
00002 *
00003 *  DO NOT EDIT THIS FILE   (rsstats-opts.c)
00004 *
00005 *  It has been AutoGen-ed
00006 *  From the definitions    rsstats-opts.def
00007 *  and the template file   options
00008 *
00009 * Generated from AutoOpts 42:1:17 templates.
00010 *
00011 *  AutoOpts is a copyrighted work.    This source file is not encumbered
00012 *  by AutoOpts licensing, but is provided under the licensing terms chosen
00013 *  by the rsstats author or copyright holder.    AutoOpts is
00014 *  licensed under the terms of the LGPL.  The redistributable library
00015 *  ("libopts") is licensed under the terms of either the LGPL or, at the
00016 *  users discretion, the BSD license.    See the AutoOpts and/or libopts sources
00017 *  for details.
00018 *
00019 * The rsstats program is copyrighted and licensed
00020 * under the following terms:
00021 *
00022 *  Copyright (C) 2024 Francois Cerbelle, all rights reserved.
00023 *  This is free software.  It is licensed for use, modification and
00024 *  redistribution under the terms of the GNU General Public License,
00025 *  version 3 or later <http://gnu.org/licenses/gpl.html>
00026 *
00027 *  rsstats is free software:  you can redistribute it and/or modify it
00028 *  under the terms of the GNU General Public License as published by the
00029 *  Free Software Foundation, either version 3 of the License, or
00030 *  (at your option) any later version.
00031 *
00032 *  rsstats is distributed in the hope that it will be useful, but
00033 *  WITHOUT ANY WARRANTY; without even the implied warranty of
00034 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
00035 *  See the GNU General Public License for more details.
00036 *
00037 *  You should have received a copy of the GNU General Public License along
00038 *  with this program.    If not, see <http://www.gnu.org/licenses/>.
00039 */
00040
00041 #ifndef __doxygen__
00042 #define OPTION_CODE_COMPILE 1
00043 #include "rsstats-opts.h"
00044 #include <sys/types.h>
00045 #include <sys/stat.h>
00046
00047 #include <errno.h>
00048 #include <fcntl.h>
00049 #include <limits.h>
00050 #include <stdio.h>
00051 #include <stdlib.h>
00052 #include <string.h>
00053 #include <unistd.h>
00054
00055 #ifdef  __cplusplus
00056 extern "C" {
00057 #endif
00058 extern FILE * option_usage_fp;
00059 #define zCopyright      (rsstats_opt_strs+0)
00060 #define zLicenseDescrip (rsstats_opt_strs+258)
00061
00062
00063 #ifndef NULL
00064 #  define NULL 0
00065 #endif
00066
00070 static char const rsstats_opt_strs[1873] =
00071 /*     0 */ "rsstats 0.0.1\n"
00072             "Copyright (C) 2024 Francois Cerbelle, all rights reserved.\n"
00073             "This is free software.  It is licensed for use, modification and\n"
00074             "redistribution under the terms of the GNU General Public License,\n"
00075             "version 3 or later <http://gnu.org/licenses/gpl.html>\n\0"
00076 /*   258 */ "rsstats is free software:  you can redistribute it and/or modify it under\n"
00077             "the terms of the GNU General Public License as published by the Free\n"
00078             "Software Foundation, either version 3 of the License, or (at your option)\n"
00079             "any later version.\n\n"
00080             "rsstats is distributed in the hope that it will be useful, but WITHOUT ANY\n"
00081             "WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS\n"
00082             "FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more\n"
00083             "details.\n\n"
00084             "You should have received a copy of the GNU General Public License along\n"
00085             "with this program.    If not, see <http://www.gnu.org/licenses/>.\n\0"
```

```
00086 /*   861 */ "input CSV file (default:  clusterdef.csv)\0"
00087 /*   902 */ "INPUT\0"
00088 /*   908 */ "input\0"
00089 /*   914 */ "clusterdef.csv\0"
00090 /*   929 */ "output CVS filename for nodes information (default:  rsstats.csv)\0"
00091 /*   994 */ "OUTPUT\0"
00092 /*  1001 */ "output\0"
00093 /*  1008 */ "rsstats.csv\0"
00094 /*  1020 */ "comma separated list of clusternames to query (default:  all)\0"
00095 /*  1081 */ "CLUSTERS\0"
00096 /*  1090 */ "clusters\0"
00097 /*  1099 */ "all\0"
00098 /*  1103 */ "Comma separated list of reports to generate (default:  all)\0"
00099 /*  1162 */ "REPORTS\0"
00100 /*  1170 */ "reports\0"
00101 /*  1178 */ "display extended usage information and exit\0"
00102 /*  1222 */ "help\0"
00103 /*  1227 */ "extended usage information passed thru pager\0"
00104 /*  1272 */ "more-help\0"
00105 /*  1282 */ "output version information and exit\0"
00106 /*  1318 */ "version\0"
00107 /*  1326 */ "save the option state to a config file\0"
00108 /*  1365 */ "save-opts\0"
00109 /*  1375 */ "load options from a config file\0"
00110 /*  1407 */ "LOAD_OPTS\0"
00111 /*  1417 */ "no-load-opts\0"
00112 /*  1430 */ "no\0"
00113 /*  1433 */ "RSSTATS\0"
00114 /*  1441 */ "rsstats - Redis Enterprise Software cluster statistic extraction\n"
00115             "Usage:    %s [ -<flag> [<val>] | --<name>[{=| }<val>] ]...\n\0"
00116 /*  1564 */ "$HOME\0"
00117 /*  1570 */ ".rsstatsrc\0"
00118 /*  1581 */ "francois@cerbelle.net\0"
00119 /*  1603 */ "additional information given whenever the usage routine is invoked.\n\0"
00120 /*  1672 */ "This string is added to the usage output when the HELP option is selected.\n"
00121             "The contents of the file 'rsstats.details' is added to the usage output\n"
00122             "when the MORE-HELP option is selected.\n\0"
00123 /*  1859 */ "rsstats 0.0.1";
00124
00129 #define INPUT_DESC      (rsstats_opt_strs+861)
00131 #define INPUT_NAME      (rsstats_opt_strs+902)
00133 #define INPUT_name      (rsstats_opt_strs+908)
00135 #define INPUT_DFT_ARG   (rsstats_opt_strs+914)
00137 #define INPUT_FLAGS     (OPTST_DISABLED \
00138 | OPTST_SET_ARGTYPE(OPARG_TYPE_FILE))
00139
00144 #define OUTPUT_DESC      (rsstats_opt_strs+929)
00146 #define OUTPUT_NAME      (rsstats_opt_strs+994)
00148 #define OUTPUT_name      (rsstats_opt_strs+1001)
00150 #define OUTPUT_DFT_ARG   (rsstats_opt_strs+1008)
00152 #define OUTPUT_FLAGS     (OPTST_DISABLED \
00153 | OPTST_SET_ARGTYPE(OPARG_TYPE_FILE))
00154
00159 #define CLUSTERS_DESC      (rsstats_opt_strs+1020)
00161 #define CLUSTERS_NAME      (rsstats_opt_strs+1081)
00163 #define CLUSTERS_name      (rsstats_opt_strs+1090)
00165 #define CLUSTERS_DFT_ARG   (rsstats_opt_strs+1099)
00167 #define CLUSTERS_FLAGS     (OPTST_DISABLED \
00168 | OPTST_SET_ARGTYPE(OPARG_TYPE_STRING))
00169
00174 #define REPORTS_DESC      (rsstats_opt_strs+1103)
00176 #define REPORTS_NAME      (rsstats_opt_strs+1162)
00178 #define REPORTS_name      (rsstats_opt_strs+1170)
00180 #define REPORTS_DFT_ARG   (NULL)
00181 #define ReportsCookieBits            VOIDP(REPORTS_BDBS|REPORTS_CLUSTER)
00183 #define REPORTS_FLAGS     (OPTST_DISABLED \
00184 | OPTST_SET_ARGTYPE(OPARG_TYPE_MEMBERSHIP))
00185
00186 /*
00187 *  Help/More_Help/Version option descriptions:
00188 */
00189 #define HELP_DESC       (rsstats_opt_strs+1178)
00190 #define HELP_name       (rsstats_opt_strs+1222)
00191 #ifdef HAVE_WORKING_FORK
00192 #define MORE_HELP_DESC  (rsstats_opt_strs+1227)
00193 #define MORE_HELP_name  (rsstats_opt_strs+1272)
00194 #define MORE_HELP_FLAGS (OPTST_IMM | OPTST_NO_INIT)
00195 #else
00196 #define MORE_HELP_DESC  HELP_DESC
00197 #define MORE_HELP_name  HELP_name
00198 #define MORE_HELP_FLAGS (OPTST_OMITTED | OPTST_NO_INIT)
00199 #endif
00200 #ifdef NO_OPTIONAL_OPT_ARGS
00201 #  define VER_FLAGS     (OPTST_IMM | OPTST_NO_INIT)
00202 #else
00203 #  define VER_FLAGS     (OPTST_SET_ARGTYPE(OPARG_TYPE_STRING) | \
00204 OPTST_ARG_OPTIONAL | OPTST_IMM | OPTST_NO_INIT)
```

```
00205 #endif
00206 #define VER_DESC        (rsstats_opt_strs+1282)
00207 #define VER_name        (rsstats_opt_strs+1318)
00208 #define SAVE_OPTS_DESC  (rsstats_opt_strs+1326)
00209 #define SAVE_OPTS_name  (rsstats_opt_strs+1365)
00210 #define LOAD_OPTS_DESC     (rsstats_opt_strs+1375)
00211 #define LOAD_OPTS_NAME     (rsstats_opt_strs+1407)
00212 #define NO_LOAD_OPTS_name  (rsstats_opt_strs+1417)
00213 #define LOAD_OPTS_pfx      (rsstats_opt_strs+1430)
00214 #define LOAD_OPTS_name     (NO_LOAD_OPTS_name + 3)
00218 extern tOptProc
00219     optionBooleanVal,   optionNestedVal,    optionNumericVal,
00220     optionPagedUsage,   optionPrintVersion, optionResetOpt,
00221     optionStackArg,     optionTimeDate,     optionTimeVal,
00222     optionUnstackArg,   optionVendorOption;
00223 static tOptProc
00224     doOptInput, doOptOutput, doOptReports, doUsageOpt;
00225 #define VER_PROC        optionPrintVersion
00226
00227 /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
00233 static tOptDesc optDesc[OPTION_CT] = {
00234   {  /* entry idx, value */ 0, VALUE_OPT_INPUT,
00235      /* equiv idx, value */ 0, VALUE_OPT_INPUT,
00236      /* equivalenced to  */ NO_EQUIVALENT,
00237      /* min, max, act ct */ 0, 1, 0,
00238      /* opt state flags  */ INPUT_FLAGS, 0,
00239      /* last opt argumnt */ { INPUT_DFT_ARG },
00240      /* arg list/cookie  */ NULL,
00241      /* must/cannot opts */ NULL, NULL,
00242      /* option proc      */ doOptInput,
00243      /* desc, NAME, name */ INPUT_DESC, INPUT_NAME, INPUT_name,
00244      /* disablement strs */ NULL, NULL },
00245
00246   {  /* entry idx, value */ 1, VALUE_OPT_OUTPUT,
00247      /* equiv idx, value */ 1, VALUE_OPT_OUTPUT,
00248      /* equivalenced to  */ NO_EQUIVALENT,
00249      /* min, max, act ct */ 0, 1, 0,
00250      /* opt state flags  */ OUTPUT_FLAGS, 0,
00251      /* last opt argumnt */ { OUTPUT_DFT_ARG },
00252      /* arg list/cookie  */ NULL,
00253      /* must/cannot opts */ NULL, NULL,
00254      /* option proc      */ doOptOutput,
00255      /* desc, NAME, name */ OUTPUT_DESC, OUTPUT_NAME, OUTPUT_name,
00256      /* disablement strs */ NULL, NULL },
00257
00258   {  /* entry idx, value */ 2, VALUE_OPT_CLUSTERS,
00259      /* equiv idx, value */ 2, VALUE_OPT_CLUSTERS,
00260      /* equivalenced to  */ NO_EQUIVALENT,
00261      /* min, max, act ct */ 0, 1, 0,
00262      /* opt state flags  */ CLUSTERS_FLAGS, 0,
00263      /* last opt argumnt */ { CLUSTERS_DFT_ARG },
00264      /* arg list/cookie  */ NULL,
00265      /* must/cannot opts */ NULL, NULL,
00266      /* option proc      */ NULL,
00267      /* desc, NAME, name */ CLUSTERS_DESC, CLUSTERS_NAME, CLUSTERS_name,
00268      /* disablement strs */ NULL, NULL },
00269
00270   {  /* entry idx, value */ 3, VALUE_OPT_REPORTS,
00271      /* equiv idx, value */ 3, VALUE_OPT_REPORTS,
00272      /* equivalenced to  */ NO_EQUIVALENT,
00273      /* min, max, act ct */ 0, NOLIMIT, 0,
00274      /* opt state flags  */ REPORTS_FLAGS, 0,
00275      /* last opt argumnt */ { REPORTS_DFT_ARG },
00276      /* arg list/cookie  */ ReportsCookieBits,
00277      /* must/cannot opts */ NULL, NULL,
00278      /* option proc      */ doOptReports,
00279      /* desc, NAME, name */ REPORTS_DESC, REPORTS_NAME, REPORTS_name,
00280      /* disablement strs */ NULL, NULL },
00281
00282   {  /* entry idx, value */ INDEX_OPT_VERSION, VALUE_OPT_VERSION,
00283      /* equiv idx value  */ NO_EQUIVALENT, VALUE_OPT_VERSION,
00284      /* equivalenced to  */ NO_EQUIVALENT,
00285      /* min, max, act ct */ 0, 1, 0,
00286      /* opt state flags  */ VER_FLAGS, AOUSE_VERSION,
00287      /* last opt argumnt */ { NULL },
00288      /* arg list/cookie  */ NULL,
00289      /* must/cannot opts */ NULL, NULL,
00290      /* option proc      */ VER_PROC,
00291      /* desc, NAME, name */ VER_DESC, NULL, VER_name,
00292      /* disablement strs */ NULL, NULL },
00293
00294
00295
00296   {  /* entry idx, value */ INDEX_OPT_HELP, VALUE_OPT_HELP,
00297      /* equiv idx value  */ NO_EQUIVALENT, VALUE_OPT_HELP,
00298      /* equivalenced to  */ NO_EQUIVALENT,
00299      /* min, max, act ct */ 0, 1, 0,
```

```
     /* opt state flags  */ OPTST_IMM | OPTST_NO_INIT, AOUSE_HELP,
     /* last opt argumnt */ { NULL },
     /* arg list/cookie  */ NULL,
     /* must/cannot opts */ NULL, NULL,
     /* option proc      */ doUsageOpt,
     /* desc, NAME, name */ HELP_DESC, NULL, HELP_name,
     /* disablement strs */ NULL, NULL },

  {  /* entry idx, value */ INDEX_OPT_MORE_HELP, VALUE_OPT_MORE_HELP,
     /* equiv idx value  */ NO_EQUIVALENT, VALUE_OPT_MORE_HELP,
     /* equivalenced to  */ NO_EQUIVALENT,
     /* min, max, act ct */ 0, 1, 0,
     /* opt state flags  */ MORE_HELP_FLAGS, AOUSE_MORE_HELP,
     /* last opt argumnt */ { NULL },
     /* arg list/cookie  */ NULL,
     /* must/cannot opts */ NULL,  NULL,
     /* option proc      */ optionPagedUsage,
     /* desc, NAME, name */ MORE_HELP_DESC, NULL, MORE_HELP_name,
     /* disablement strs */ NULL, NULL },

  {  /* entry idx, value */ INDEX_OPT_SAVE_OPTS, VALUE_OPT_SAVE_OPTS,
     /* equiv idx value  */ NO_EQUIVALENT, VALUE_OPT_SAVE_OPTS,
     /* equivalenced to  */ NO_EQUIVALENT,
     /* min, max, act ct */ 0, 1, 0,
     /* opt state flags  */ OPTST_SET_ARGTYPE(OPARG_TYPE_STRING)
                       | OPTST_ARG_OPTIONAL | OPTST_NO_INIT, AOUSE_SAVE_OPTS,
     /* last opt argumnt */ { NULL },
     /* arg list/cookie  */ NULL,
     /* must/cannot opts */ NULL,  NULL,
     /* option proc      */ NULL,
     /* desc, NAME, name */ SAVE_OPTS_DESC, NULL, SAVE_OPTS_name,
     /* disablement strs */ NULL, NULL },

  {  /* entry idx, value */ INDEX_OPT_LOAD_OPTS, VALUE_OPT_LOAD_OPTS,
     /* equiv idx value  */ NO_EQUIVALENT, VALUE_OPT_LOAD_OPTS,
     /* equivalenced to  */ NO_EQUIVALENT,
     /* min, max, act ct */ 0, NOLIMIT, 0,
     /* opt state flags  */ OPTST_SET_ARGTYPE(OPARG_TYPE_STRING)
                 | OPTST_DISABLE_IMM, AOUSE_LOAD_OPTS,
     /* last opt argumnt */ { NULL },
     /* arg list/cookie  */ NULL,
     /* must/cannot opts */ NULL, NULL,
     /* option proc      */ optionLoadOpt,
     /* desc, NAME, name */ LOAD_OPTS_DESC, LOAD_OPTS_NAME, LOAD_OPTS_name,
     /* disablement strs */ NO_LOAD_OPTS_name, LOAD_OPTS_pfx }
};


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#define zPROGNAME       (rsstats_opt_strs+1433)
#define zUsageTitle     (rsstats_opt_strs+1441)
#define zRcName         (rsstats_opt_strs+1570)
static char const * const apzHomeList[2] = {
    rsstats_opt_strs+1564,
    NULL };
#define zBugsAddr       (rsstats_opt_strs+1581)
#define zExplain        (rsstats_opt_strs+1603)
#define zDetail         (rsstats_opt_strs+1672)
#define zFullVersion    (rsstats_opt_strs+1859)
/* extracted from optcode.tlib near line 342 */

#if defined(ENABLE_NLS)
# define OPTPROC_BASE OPTPROC_TRANSLATE
  static tOptionXlateProc translate_option_strings;
#else
# define OPTPROC_BASE OPTPROC_NONE
# define translate_option_strings NULL
#endif /* ENABLE_NLS */

#define rsstats_full_usage (NULL)
#define rsstats_short_usage (NULL)

#endif /* not defined __doxygen__ */

/*
*  Create the static procedure(s) declared above.
*/
static void
doUsageOpt(tOptions * opts, tOptDesc * od)
{
    int ex_code;
    ex_code = RSSTATS_EXIT_SUCCESS;
    optionUsage(&rsstatsOptions, ex_code);
    /* NOTREACHED */
    exit(RSSTATS_EXIT_FAILURE);
    (void)opts;
    (void)od;
```

```
00402 }
00403
00404 /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
00411 static void
00412 doOptInput(tOptions* pOptions, tOptDesc* pOptDesc)
00413 {
00414     static teOptFileType const  type =
00415         FTYPE_MODE_MAY_EXIST + FTYPE_MODE_NO_OPEN;
00416     static tuFileMode           mode;
00417 #ifndef O_CLOEXEC
00418 #  define O_CLOEXEC 0
00419 #endif
00420     mode.file_flags = O_CLOEXEC;
00421
00422     /*
00423 * This function handles special invalid values for "pOptions"
00424 */
00425     optionFileCheck(pOptions, pOptDesc, type, mode);
00426 }
00427
00428 /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
00435 static void
00436 doOptOutput(tOptions* pOptions, tOptDesc* pOptDesc)
00437 {
00438     static teOptFileType const  type =
00439         FTYPE_MODE_MAY_EXIST + FTYPE_MODE_NO_OPEN;
00440     static tuFileMode           mode;
00441 #ifndef O_CLOEXEC
00442 #  define O_CLOEXEC 0
00443 #endif
00444     mode.file_flags = O_CLOEXEC;
00445
00446     /*
00447 * This function handles special invalid values for "pOptions"
00448 */
00449     optionFileCheck(pOptions, pOptDesc, type, mode);
00450 }
00451
00452 /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
00459 static void
00460 doOptReports(tOptions* pOptions, tOptDesc* pOptDesc)
00461 {
00462
00463 /* extracted from optmain.tlib near line 1007 */
00464     static char const * const names[2] = {
00465         "bdbs",     "cluster"
00466     };
00467     /*
00468 * This function handles special invalid values for "pOptions"
00469 */
00470     optionSetMembers(pOptions, pOptDesc, names, 2);
00471 }
00472 /* extracted from optmain.tlib near line 1250 */
00473
00477 #ifndef  PKGDATADIR
00478 # define PKGDATADIR ""
00479 #endif
00480
00485 #ifndef  WITH_PACKAGER
00486 # define rsstats_packager_info NULL
00487 #else
00489 static char const rsstats_packager_info[] =
00490     "Packaged by " WITH_PACKAGER
00491
00492 # ifdef WITH_PACKAGER_VERSION
00493         " ("WITH_PACKAGER_VERSION")"
00494 # endif
00495
00496 # ifdef WITH_PACKAGER_BUG_REPORTS
00497     "\nReport rsstats bugs to " WITH_PACKAGER_BUG_REPORTS
00498 # endif
00499     "\n";
00500 #endif
00501 #ifndef __doxygen__
00502
00503 #endif /* __doxygen__ */
00508 tOptions rsstatsOptions = {
00509     OPTIONS_STRUCT_VERSION,
00510     0, NULL,                    /* original argc + argv    */
00511     ( OPTPROC_BASE
00512     + OPTPROC_ERRSTOP
00513     + OPTPROC_SHORTOPT
00514     + OPTPROC_LONGOPT
00515     + OPTPROC_NO_REQ_OPT
00516     + OPTPROC_ENVIRON
00517     + OPTPROC_NO_ARGS
00518     + OPTPROC_GNUUSAGE ),
```

```
00519     0, NULL,                    /* current option index, current option */
00520     NULL,         NULL,         zPROGNAME,
00521     zRcName,      zCopyright,   zLicenseDescrip,
00522     zFullVersion, apzHomeList,  zUsageTitle,
00523     zExplain,     zDetail,      optDesc,
00524     zBugsAddr,                  /* address to send bugs to */
00525     NULL, NULL,                 /* extensions/saved state  */
00526     optionUsage, /* usage procedure */
00527     translate_option_strings,   /* translation procedure */
00528     /*
00529 *  Indexes to special options
00530 */
00531     { INDEX_OPT_MORE_HELP, /* more-help option index */
00532       INDEX_OPT_SAVE_OPTS, /* save option index */
00533       NO_EQUIVALENT, /* '-#' option index */
00534       NO_EQUIVALENT /* index of default opt */
00535     },
00536     9 /* full option count */, 4 /* user option count */,
00537     rsstats_full_usage, rsstats_short_usage,
00538     NULL, NULL,
00539     PKGDATADIR, rsstats_packager_info
00540 };
00541
00542 #if ENABLE_NLS
00548 #include <stdio.h>
00549 #include <stdlib.h>
00550 #include <string.h>
00551 #include <unistd.h>
00552 #ifdef HAVE_DCGETTEXT
00553 # include <gettext.h>
00554 #endif
00555 #include <autoopts/usage-txt.h>
00556
00557 static char * AO_gettext(char const * pz);
00558 static void   coerce_it(void ** s);
00559
00570 static char *
00571 AO_gettext(char const * pz)
00572 {
00573     char * res;
00574     if (pz == NULL)
00575         return NULL;
00576 #ifdef HAVE_DCGETTEXT
00577     /*
00578 * While processing the option_xlateable_txt data, try to use the
00579 * "libopts" domain.    Once we switch to the option descriptor data,
00580 * do *not* use that domain.
00581 */
00582     if (option_xlateable_txt.field_ct != 0) {
00583         res = dgettext("libopts", pz);
00584         if (res == pz)
00585             res = (char *)VOIDP(_(pz));
00586     } else
00587         res = (char *)VOIDP(_(pz));
00588 #else
00589     res = (char *)VOIDP(_(pz));
00590 #endif
00591     if (res == pz)
00592         return res;
00593     res = strdup(res);
00594     if (res == NULL) {
00595         fputs(_("No memory for duping translated strings\n"), stderr);
00596         exit(RSSTATS_EXIT_FAILURE);
00597     }
00598     return res;
00599 }
00600
00605 static void coerce_it(void ** s) { *s = AO_gettext(*s);
00606 }
00607
00612 static void
00613 translate_option_strings(void)
00614 {
00615     tOptions * const opts = &rsstatsOptions;
00616
00617     /*
00618 *  Guard against re-translation.    It won't work.    The strings will have
00619 *  been changed by the first pass through this code.    One shot only.
00620 */
00621     if (option_xlateable_txt.field_ct != 0) {
00622         /*
00623 *  Do the translations.    The first pointer follows the field count
00624 *  field.    The field count field is the size of a pointer.
00625 */
00626         char ** ppz = (char**)VOIDP(&(option_xlateable_txt));
00627         int     ix  = option_xlateable_txt.field_ct;
00628
```

```
00629         do {
00630             ppz++; /* skip over field_ct */
00631             *ppz = AO_gettext(*ppz);
00632         } while (--ix > 0);
00633         /* prevent re-translation and disable "libopts" domain lookup */
00634         option_xlateable_txt.field_ct = 0;
00635 
00636         coerce_it(VOIDP(&(opts->pzCopyright)));
00637         coerce_it(VOIDP(&(opts->pzCopyNotice)));
00638         coerce_it(VOIDP(&(opts->pzFullVersion)));
00639         coerce_it(VOIDP(&(opts->pzUsageTitle)));
00640         coerce_it(VOIDP(&(opts->pzExplain)));
00641         coerce_it(VOIDP(&(opts->pzDetail)));
00642         {
00643             tOptDesc * od = opts->pOptDesc;
00644             for (ix = opts->optCt; ix > 0; ix--, od++)
00645                 coerce_it(VOIDP(&(od->pzText)));
00646         }
00647     }
00648 }
00649 #endif /* ENABLE_NLS */
00650 
00651 #ifdef DO_NOT_COMPILE_THIS_CODE_IT_IS_FOR_GETTEXT
00653 static void bogus_function(void) {
00654   /* TRANSLATORS:
00655 
00656 The following dummy function was crated solely so that xgettext can
00657 extract the correct strings.    These strings are actually referenced
00658 by a field name in the rsstatsOptions structure noted in the
00659 comments below.    The literal text is defined in rsstats_opt_strs.
00660 
00661 NOTE: the strings below are segmented with respect to the source string
00662 rsstats_opt_strs.    The strings above are handed off for translation
00663 at run time a paragraph at a time.    Consequently, they are presented here
00664 for translation a paragraph at a time.
00665 
00666 ALSO: often the description for an option will reference another option
00667 by name.    These are set off with apostrophe quotes (I hope).    Do not
00668 translate option names.
00669 */
00670   /* referenced via rsstatsOptions.pzCopyright */
00671   puts(_("rsstats 0.0.1\n\
00672 Copyright (C) 2024 Francois Cerbelle, all rights reserved.\n\
00673 This is free software.  It is licensed for use, modification and\n\
00674 redistribution under the terms of the GNU General Public License,\n\
00675 version 3 or later <http://gnu.org/licenses/gpl.html>\n"));
00676 
00677   /* referenced via rsstatsOptions.pzCopyNotice */
00678   puts(_("rsstats is free software:  you can redistribute it and/or modify it under\n\
00679 the terms of the GNU General Public License as published by the Free\n\
00680 Software Foundation, either version 3 of the License, or (at your option)\n\
00681 any later version.\n\n"));
00682   puts(_("rsstats is distributed in the hope that it will be useful, but WITHOUT ANY\n\
00683 WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS\n\
00684 FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more\n\
00685 details.\n\n"));
00686   puts(_("You should have received a copy of the GNU General Public License along\n\
00687 with this program.    If not, see <http://www.gnu.org/licenses/>.\n"));
00688 
00689   /* referenced via rsstatsOptions.pOptDesc->pzText */
00690   puts(_("input CSV file (default:  clusterdef.csv)"));
00691 
00692   /* referenced via rsstatsOptions.pOptDesc->pzText */
00693   puts(_("output CVS filename for nodes information (default:  rsstats.csv)"));
00694 
00695   /* referenced via rsstatsOptions.pOptDesc->pzText */
00696   puts(_("comma separated list of clusternames to query (default:  all)"));
00697 
00698   /* referenced via rsstatsOptions.pOptDesc->pzText */
00699   puts(_("Comma separated list of reports to generate (default:  all)"));
00700 
00701   /* referenced via rsstatsOptions.pOptDesc->pzText */
00702   puts(_("display extended usage information and exit"));
00703 
00704   /* referenced via rsstatsOptions.pOptDesc->pzText */
00705   puts(_("extended usage information passed thru pager"));
00706 
00707   /* referenced via rsstatsOptions.pOptDesc->pzText */
00708   puts(_("output version information and exit"));
00709 
00710   /* referenced via rsstatsOptions.pOptDesc->pzText */
00711   puts(_("save the option state to a config file"));
00712 
00713   /* referenced via rsstatsOptions.pOptDesc->pzText */
00714   puts(_("load options from a config file"));
00715 
00716   /* referenced via rsstatsOptions.pzUsageTitle */
```

```
00717   puts(_("rsstats - Redis Enterprise Software cluster statistic extraction\n\
00718 Usage:    %s [ -<flag> [<val>] | --<name>[{=| }<val>] ]...\n"));
00719
00720   /* referenced via rsstatsOptions.pzExplain */
00721   puts(_("additional information given whenever the usage routine is invoked.\n"));
00722
00723   /* referenced via rsstatsOptions.pzDetail */
00724   puts(_("This string is added to the usage output when the HELP option is selected.\n\
00725 The contents of the file 'rsstats.details' is added to the usage output\n\
00726 when the MORE-HELP option is selected.\n"));
00727
00728   /* referenced via rsstatsOptions.pzFullVersion */
00729   puts(_("rsstats 0.0.1"));
00730
00731   /* referenced via rsstatsOptions.pzFullUsage */
00732   puts(_("<<NOT-FOUND>>"));
00733
00734   /* referenced via rsstatsOptions.pzShortUsage */
00735   puts(_("<<NOT-FOUND>>"));
00736   /* LIBOPTS-MESSAGES: */
00737 #line 67 "../autoopts.c"
00738   puts(_("allocation of %d bytes failed\n"));
00739 #line 89 "../autoopts.c"
00740   puts(_("allocation of %d bytes failed\n"));
00741 #line 48 "../init.c"
00742   puts(_("AutoOpts function called without option descriptor\n"));
00743 #line 81 "../init.c"
00744   puts(_("\tThis exceeds the compiled library version:    "));
00745 #line 79 "../init.c"
00746   puts(_("Automated Options Processing Error!\n"
00747         "\t%s called AutoOpts function with structure version %d:%d:%d.\n"));
00748 #line 78 "../autoopts.c"
00749   puts(_("realloc of %d bytes at 0x%p failed\n"));
00750 #line 83 "../init.c"
00751   puts(_("\tThis is less than the minimum library version:    "));
00752 #line 121 "../version.c"
00753   puts(_("Automated Options version %s\n"
00754         "\tCopyright (C) 1999-2017 by Bruce Korb - all rights reserved\n"));
00755 #line 49 "../makeshell.c"
00756   puts(_("(AutoOpts bug):    %s.\n"));
00757 #line 90 "../reset.c"
00758   puts(_("optionResetOpt() called, but reset-option not configured"));
00759 #line 241 "../usage.c"
00760   puts(_("could not locate the 'help' option"));
00761 #line 330 "../autoopts.c"
00762   puts(_("optionProcess() was called with invalid data"));
00763 #line 697 "../usage.c"
00764   puts(_("invalid argument type specified"));
00765 #line 568 "../find.c"
00766   puts(_("defaulted to option with optional arg"));
00767 #line 76 "../alias.c"
00768   puts(_("aliasing option is out of range."));
00769 #line 210 "../enum.c"
00770   puts(_("%s error:    the keyword '%s' is ambiguous for %s\n"));
00771 #line 78 "../find.c"
00772   puts(_("  The following options match:\n"));
00773 #line 263 "../find.c"
00774   puts(_("%s:  ambiguous option name:  %s (matches %d options)\n"));
00775 #line 161 "../check.c"
00776   puts(_("%s:  Command line arguments required\n"));
00777 #line 43 "../alias.c"
00778   puts(_("%d %s%s options allowed\n"));
00779 #line 56 "../makeshell.c"
00780   puts(_("%s error %d (%s) calling %s for '%s'\n"));
00781 #line 268 "../makeshell.c"
00782   puts(_("interprocess pipe"));
00783 #line 171 "../version.c"
00784   puts(_("error:  version option argument '%c' invalid.    Use:\n"
00785         "\t'v' - version only\n"
00786         "\t'c' - version and copyright\n"
00787         "\t'n' - version and full copyright notice\n"));
00788 #line 58 "../check.c"
00789   puts(_("%s error:    the '%s' and '%s' options conflict\n"));
00790 #line 187 "../find.c"
00791   puts(_("%s:  The '%s' option has been disabled."));
00792 #line 400 "../find.c"
00793   puts(_("%s:  The '%s' option has been disabled."));
00794 #line 38 "../alias.c"
00795   puts(_("-equivalence"));
00796 #line 439 "../find.c"
00797   puts(_("%s:  illegal option -- %c\n"));
00798 #line 110 "../reset.c"
00799   puts(_("%s:  illegal option -- %c\n"));
00800 #line 241 "../find.c"
00801   puts(_("%s:  illegal option -- %s\n"));
00802 #line 740 "../find.c"
00803   puts(_("%s:  illegal option -- %s\n"));
```

```
00804 #line 118 "../reset.c"
00805   puts(_("%s:  illegal option -- %s\n"));
00806 #line 305 "../find.c"
00807   puts(_("%s:  unknown vendor extension option -- %s\n"));
00808 #line 135 "../enum.c"
00809   puts(_("  or an integer from %d through %d\n"));
00810 #line 145 "../enum.c"
00811   puts(_("  or an integer from %d through %d\n"));
00812 #line 696 "../usage.c"
00813   puts(_("%s error:    invalid option descriptor for %s\n"));
00814 #line 1030 "../usage.c"
00815   puts(_("%s error:    invalid option descriptor for %s\n"));
00816 #line 355 "../find.c"
00817   puts(_("%s:  invalid option name:  %s\n"));
00818 #line 497 "../find.c"
00819   puts(_("%s:  The '%s' option requires an argument.\n"));
00820 #line 150 "../autoopts.c"
00821   puts(_("(AutoOpts bug):    Equivalenced option '%s' was equivalenced to both\n"
00822         "\t'%s' and '%s'."));
00823 #line 94 "../check.c"
00824   puts(_("%s error:    The %s option is required\n"));
00825 #line 602 "../find.c"
00826   puts(_("%s:  The '%s' option cannot have an argument.\n"));
00827 #line 151 "../check.c"
00828   puts(_("%s:  Command line arguments are not allowed.\n"));
00829 #line 568 "../save.c"
00830   puts(_("error %d (%s) creating %s\n"));
00831 #line 210 "../enum.c"
00832   puts(_("%s error:    '%s' does not match any %s keywords.\n"));
00833 #line 93 "../reset.c"
00834   puts(_("%s error:  The '%s' option requires an argument.\n"));
00835 #line 122 "../save.c"
00836   puts(_("error %d (%s) stat-ing %s\n"));
00837 #line 175 "../save.c"
00838   puts(_("error %d (%s) stat-ing %s\n"));
00839 #line 143 "../restore.c"
00840   puts(_("%s error:  no saved option state\n"));
00841 #line 225 "../autoopts.c"
00842   puts(_("'%s' is not a command line option.\n"));
00843 #line 113 "../time.c"
00844   puts(_("%s error:    '%s' is not a recognizable date/time.\n"));
00845 #line 50 "../time.c"
00846   puts(_("%s error:    '%s' is not a recognizable time duration.\n"));
00847 #line 92 "../check.c"
00848   puts(_("%s error:    The %s option must appear %d times.\n"));
00849 #line 165 "../numeric.c"
00850   puts(_("%s error:    '%s' is not a recognizable number.\n"));
00851 #line 176 "../enum.c"
00852   puts(_("%s error:    %s exceeds %s keyword count\n"));
00853 #line 279 "../usage.c"
00854   puts(_("Try '%s %s' for more information.\n"));
00855 #line 45 "../alias.c"
00856   puts(_("one %s%s option allowed\n"));
00857 #line 170 "../makeshell.c"
00858   puts(_("standard output"));
00859 #line 905 "../makeshell.c"
00860   puts(_("standard output"));
00861 #line 223 "../usage.c"
00862   puts(_("standard output"));
00863 #line 364 "../usage.c"
00864   puts(_("standard output"));
00865 #line 574 "../usage.c"
00866   puts(_("standard output"));
00867 #line 178 "../version.c"
00868   puts(_("standard output"));
00869 #line 223 "../usage.c"
00870   puts(_("standard error"));
00871 #line 364 "../usage.c"
00872   puts(_("standard error"));
00873 #line 574 "../usage.c"
00874   puts(_("standard error"));
00875 #line 178 "../version.c"
00876   puts(_("standard error"));
00877 #line 170 "../makeshell.c"
00878   puts(_("write"));
00879 #line 905 "../makeshell.c"
00880   puts(_("write"));
00881 #line 222 "../usage.c"
00882   puts(_("write"));
00883 #line 363 "../usage.c"
00884   puts(_("write"));
00885 #line 573 "../usage.c"
00886   puts(_("write"));
00887 #line 177 "../version.c"
00888   puts(_("write"));
00889 #line 60 "../numeric.c"
00890   puts(_("%s error:    %s option value %ld is out of range.\n"));
```

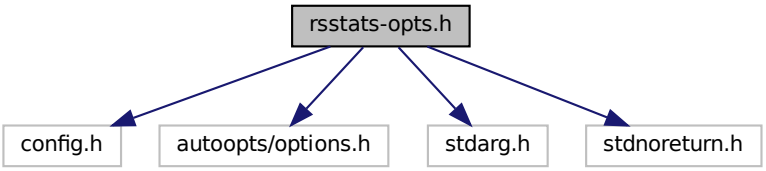
```
00891 #line 44 "../check.c"
00892   puts(_("%s error:    %s option requires the %s option\n"));
00893 #line 121 "../save.c"
00894   puts(_("%s warning:    cannot save options - %s not regular file\n"));
00895 #line 174 "../save.c"
00896   puts(_("%s warning:    cannot save options - %s not regular file\n"));
00897 #line 193 "../save.c"
00898   puts(_("%s warning:    cannot save options - %s not regular file\n"));
00899 #line 567 "../save.c"
00900   puts(_("%s warning:    cannot save options - %s not regular file\n"));
00901   /* END-LIBOPTS-MESSAGES */
00902
00903   /* USAGE-TEXT: */
00904 #line 822 "../usage.c"
00905   puts(_("\t\t\t\t- an alternate for '%s'\n"));
00906 #line 1097 "../usage.c"
00907   puts(_("Version, usage and configuration options:"));
00908 #line 873 "../usage.c"
00909   puts(_("\t\t\t\t- default option for unnamed options\n"));
00910 #line 786 "../usage.c"
00911   puts(_("\t\t\t\t- disabled as '--%s'\n"));
00912 #line 1066 "../usage.c"
00913   puts(_(" --- %-14s %s\n"));
00914 #line 1064 "../usage.c"
00915   puts(_("This option has been disabled"));
00916 #line 813 "../usage.c"
00917   puts(_("\t\t\t\t- enabled by default\n"));
00918 #line 40 "../alias.c"
00919   puts(_("%s error:    only "));
00920 #line 1143 "../usage.c"
00921   puts(_(" - examining environment variables named %s_*\n"));
00922 #line 168 "../file.c"
00923   puts(_("\t\t\t\t- file must not pre-exist\n"));
00924 #line 172 "../file.c"
00925   puts(_("\t\t\t\t- file must pre-exist\n"));
00926 #line 329 "../usage.c"
00927   puts(_("Options are specified by doubled hyphens and their name or by a single\n"
00928          "hyphen and the flag character.\n"));
00929 #line 882 "../makeshell.c"
00930   puts(_("\n"
00931          "= = = = = = = =\n\n"
00932          "This incarnation of genshell will produce\n"
00933          "a shell script to parse the options for %s:\n\n"));
00934 #line 142 "../enum.c"
00935   puts(_("  or an integer mask with any of the lower %d bits set\n"));
00936 #line 846 "../usage.c"
00937   puts(_("\t\t\t\t- is a set membership option\n"));
00938 #line 867 "../usage.c"
00939   puts(_("\t\t\t\t- must appear between %d and %d times\n"));
00940 #line 331 "../usage.c"
00941   puts(_("Options are specified by single or double hyphens and their name.\n"));
00942 #line 853 "../usage.c"
00943   puts(_("\t\t\t\t- may appear multiple times\n"));
00944 #line 840 "../usage.c"
00945   puts(_("\t\t\t\t- may not be preset\n"));
00946 #line 1258 "../usage.c"
00947   puts(_("   Arg Option-Name    Description\n"));
00948 #line 1194 "../usage.c"
00949   puts(_("  Flg Arg Option-Name    Description\n"));
00950 #line 1252 "../usage.c"
00951   puts(_("  Flg Arg Option-Name    Description\n"));
00952 #line 1253 "../usage.c"
00953   puts(_(" %3s %s"));
00954 #line 1259 "../usage.c"
00955   puts(_(" %3s %s"));
00956 #line 336 "../usage.c"
00957   puts(_("The '-#<number>' option may omit the hash char\n"));
00958 #line 332 "../usage.c"
00959   puts(_("All arguments are named options.\n"));
00960 #line 920 "../usage.c"
00961   puts(_(" - reading file %s"));
00962 #line 358 "../usage.c"
00963   puts(_("\n"
00964          "Please send bug reports to:    <%s>\n"));
00965 #line 100 "../version.c"
00966   puts(_("\n"
00967          "Please send bug reports to:    <%s>\n"));
00968 #line 129 "../version.c"
00969   puts(_("\n"
00970          "Please send bug reports to:    <%s>\n"));
00971 #line 852 "../usage.c"
00972   puts(_("\t\t\t\t- may NOT appear - preset only\n"));
00973 #line 893 "../usage.c"
00974   puts(_("\n"
00975          "The following option preset mechanisms are supported:\n"));
00976 #line 1141 "../usage.c"
00977   puts(_("\n"
```

```
00978         "The following option preset mechanisms are supported:\n"));
00979 #line 631 "../usage.c"
00980   puts(_("prohibits these options:\n"));
00981 #line 626 "../usage.c"
00982   puts(_("prohibits the option '%s'\n"));
00983 #line 81 "../numeric.c"
00984   puts(_("%s%ld to %ld"));
00985 #line 79 "../numeric.c"
00986   puts(_("%sgreater than or equal to %ld"));
00987 #line 75 "../numeric.c"
00988   puts(_("%s%ld exactly"));
00989 #line 68 "../numeric.c"
00990   puts(_("%sit must lie in one of the ranges:\n"));
00991 #line 68 "../numeric.c"
00992   puts(_("%sit must be in the range:\n"));
00993 #line 88 "../numeric.c"
00994   puts(_(", or\n"));
00995 #line 66 "../numeric.c"
00996   puts(_("%sis scalable with a suffix:  k/K/m/M/g/G/t/T\n"));
00997 #line 77 "../numeric.c"
00998   puts(_("%sless than or equal to %ld"));
00999 #line 339 "../usage.c"
01000   puts(_("Operands and options may be intermixed.    They will be reordered.\n"));
01001 #line 601 "../usage.c"
01002   puts(_("requires the option '%s'\n"));
01003 #line 604 "../usage.c"
01004   puts(_("requires these options:\n"));
01005 #line 1270 "../usage.c"
01006   puts(_("   Arg Option-Name   Req?    Description\n"));
01007 #line 1264 "../usage.c"
01008   puts(_("  Flg Arg Option-Name   Req?    Description\n"));
01009 #line 143 "../enum.c"
01010   puts(_("or you may use a numeric representation.    Preceding these with a '!'\n"
01011         "will clear the bits, specifying 'none' will clear all bits, and 'all'\n"
01012         "will set them all.    Multiple entries may be passed as an option\n"
01013         "argument list.\n"));
01014 #line 859 "../usage.c"
01015   puts(_("\t\t\t\t- may appear up to %d times\n"));
01016 #line 52 "../enum.c"
01017   puts(_("The valid \"%s\" option keywords are:\n"));
01018 #line 1101 "../usage.c"
01019   puts(_("The next option supports vendor supported extra options:"));
01020 #line 722 "../usage.c"
01021   puts(_("These additional options are:"));
01022   /* END-USAGE-TEXT */
01023 }
01024 #endif /* uncompilable code */
01025 #ifdef  __cplusplus
01026 }
01027 #endif
01028 /* rsstats-opts.c ends here */
```

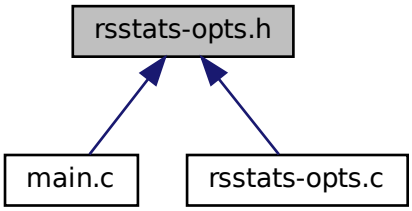
4.43 rsstats-opts.h File Reference

```
#include "config.h"
#include <autoopts/options.h>
#include <stdarg.h>
#include <stdnoreturn.h>
```

Include dependency graph for rsstats-opts.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define AO_TEMPLATE_VERSION 172033

  This file contains the programmatic interface to the Automated Options generated for the rsstats program.
- #define NOT_REACHED
- #define OPTION_CT 9

  count of all options for rsstats
- #define RSSTATS_VERSION "0.0.1"

  rsstats version
- #define RSSTATS_FULL_VERSION "rsstats 0.0.1"

  Full rsstats version text.
- #define DESC(n) (rsstatsOptions.pOptDesc[INDEX_OPT_## n])

  Interface defines for all options.
- #define HAVE_OPT(n) (! UNUSED_OPT(& DESC(n)))

  'true' if an option has been specified in any way
- #define OPT_ARG(n) (DESC(n).optArg.argString)

  The string argument to an option.
- #define STATE_OPT(n) (DESC(n).fOptState & OPTST_SET_MASK)

  Mask the option state revealing how an option was specified.
- #define COUNT_OPT(n) (DESC(n).optOccCt)

  Count of option's occurrances on the command line.
- #define ISSEL_OPT(n) (SELECTED_OPT(&DESC(n)))

  mask of OPTST_SET and OPTST_DEFINED.
- #define ISUNUSED_OPT(n) (UNUSED_OPT(& DESC(n)))

  'true' if HAVE_OPT would yield 'false'.
- #define ENABLED_OPT(n) (! DISABLED_OPT(& DESC(n)))

  'true' if OPTST_DISABLED bit not set.
- #define STACKCT_OPT(n) (((tArgList*)(DESC(n).optCookie))->useCt)

  number of stacked option arguments.
- #define STACKLST_OPT(n) (((tArgList*)(DESC(n).optCookie))->apzArgs)

  stacked argument vector.
- #define CLEAR_OPT(n)

  Reset an option.

Enumerations

- enum teOptIndex {
  INDEX_OPT_INPUT = 0 , INDEX_OPT_OUTPUT = 1 , INDEX_OPT_CLUSTERS = 2 , INDEX_OPT_REPORTS = 3 ,
  INDEX_OPT_VERSION = 4 , INDEX_OPT_HELP = 5 , INDEX_OPT_MORE_HELP = 6 , INDEX_OPT_SAVE_OPTS = 7 ,
  INDEX_OPT_LOAD_OPTS = 8 }

  Enumeration of each option type for rsstats.
- enum rsstats_exit_code_t {
  RSSTATS_EXIT_SUCCESS = 0 , RSSTATS_EXIT_FAILURE = 1 , RSSTATS_EXIT_USAGE_ERROR = 64 ,
  RSSTATS_EXIT_NO_CONFIG_INPUT = 66 ,
  RSSTATS_EXIT_LIBOPTS_FAILURE = 70 }

  Enumeration of rsstats exit codes.

- #define VALUE_OPT_INPUT 'i'

  Interface defines for specific options.
- #define VALUE_OPT_OUTPUT 'o'
- #define VALUE_OPT_CLUSTERS 'c'
- #define VALUE_OPT_REPORTS 'r'
- #define REPORTS_BDBS 0x1UL
- #define REPORTS_CLUSTER 0x2UL
- #define REPORTS_MEMBERSHIP_MASK 0x3UL
- #define OPT_VALUE_REPORTS ((uintptr_t)DESC(REPORTS).optCookie)
- #define OPT_MEMLST_REPORTS optionMemberList(&DESC(REPORTS))
- #define VALUE_OPT_HELP 'h'

  option flag (value) for help-value option
- #define VALUE_OPT_MORE_HELP 'H'

  option flag (value) for more-help-value option
- #define VALUE_OPT_VERSION 'v'

  option flag (value) for version-value option
- #define VALUE_OPT_SAVE_OPTS '>'

  option flag (value) for save-opts-value option
- #define VALUE_OPT_LOAD_OPTS '<'

  option flag (value) for load-opts-value option
- #define SET_OPT_SAVE_OPTS(a)
- #define ERRSKIP_OPTERR STMTS(rsstatsOptions.fOptSet &= ~OPTPROC_ERRSTOP)
- #define ERRSTOP_OPTERR STMTS(rsstatsOptions.fOptSet |= OPTPROC_ERRSTOP)
- #define RESTART_OPT(n)
- #define START_OPT RESTART_OPT(1)
- #define USAGE(c) (*rsstatsOptions.pUsageProc)(&rsstatsOptions, c)
- #define OPT_NO_XLAT_CFG_NAMES
- #define OPT_NO_XLAT_OPT_NAMES
- #define OPT_XLAT_CFG_NAMES
- #define OPT_XLAT_OPT_NAMES
- #define _(_s) _s
- tOptions rsstatsOptions

  The option definitions for rsstats.

4.43.1 Macro Definition Documentation

4.43.1.1 _

```
#define _(
              _s ) _s
```

Definition at line 229 of file rsstats-opts.h.

4.43.1.2 AO_TEMPLATE_VERSION

```
#define AO_TEMPLATE_VERSION 172033
```

This file contains the programmatic interface to the Automated Options generated for the rsstats program.

These macros are documented in the AutoGen info file in the "AutoOpts" chapter. Please refer to that doc for usage help. Ensure that the library used for compiling this generated header is at least as new as the version current when the header template was released (not counting patch version increments). Also ensure that the oldest tolerable version is at least as old as what was current when the header template was released.

Definition at line 60 of file rsstats-opts.h.

4.43.1.3 CLEAR_OPT

```
#define CLEAR_OPT(
              n )
```

Value:

```
STMTS( \
DESC(n).fOptState &= OPTST_PERSISTENT_MASK;   \
if ((DESC(n).fOptState & OPTST_INITENABLED) == 0) \
    DESC(n).fOptState |= OPTST_DISABLED; \
DESC(n).optCookie = NULL )
```

Reset an option.

Definition at line 124 of file rsstats-opts.h.

4.43.1.4 COUNT_OPT

```
#define COUNT_OPT(
              n ) (DESC(n).optOccCt)
```

Count of option's occurrances *on the command line*.

Definition at line 110 of file rsstats-opts.h.

4.43.1.5 DESC

```
#define DESC(
             n ) (rsstatsOptions.pOptDesc[INDEX_OPT_## n])
```

Interface defines for all options.

Replace "n" with the UPPER_CASED option name (as in the teOptIndex enumeration above). e.↩ g. HAVE_OPT(INPUT)

Definition at line 99 of file rsstats-opts.h.

4.43.1.6 ENABLED_OPT

```
#define ENABLED_OPT(
             n ) (!  DISABLED_OPT(& DESC(n)))
```

'true' if OPTST_DISABLED bit not set.

Definition at line 116 of file rsstats-opts.h.

4.43.1.7 ERRSKIP_OPTERR

```
#define ERRSKIP_OPTERR STMTS(rsstatsOptions.fOptSet &= ~OPTPROC_ERRSTOP)
```

Definition at line 171 of file rsstats-opts.h.

4.43.1.8 ERRSTOP_OPTERR

```
#define ERRSTOP_OPTERR STMTS(rsstatsOptions.fOptSet |= OPTPROC_ERRSTOP)
```

Definition at line 172 of file rsstats-opts.h.

4.43.1.9 HAVE_OPT

```
#define HAVE_OPT(
             n ) (!  UNUSED_OPT(& DESC(n)))
```

'true' if an option has been specified in any way

Definition at line 101 of file rsstats-opts.h.

4.43.1.10 ISSEL_OPT

```
#define ISSEL_OPT(
              n ) (SELECTED_OPT(&DESC(n)))
```

mask of *OPTST_SET* and *OPTST_DEFINED*.

Definition at line 112 of file rsstats-opts.h.

4.43.1.11 ISUNUSED_OPT

```
#define ISUNUSED_OPT(
              n ) (UNUSED_OPT(& DESC(n)))
```

'true' if *HAVE_OPT* would yield 'false'.

Definition at line 114 of file rsstats-opts.h.

4.43.1.12 NOT_REACHED

```
#define NOT_REACHED
```

Definition at line 70 of file rsstats-opts.h.

4.43.1.13 OPT_ARG

```
#define OPT_ARG(
              n ) (DESC(n).optArg.argString)
```

The string argument to an option.

The argument type must be "string".

Definition at line 103 of file rsstats-opts.h.

4.43.1.14 OPT_MEMLST_REPORTS

```
#define OPT_MEMLST_REPORTS optionMemberList(&DESC(REPORTS))
```

Definition at line 153 of file rsstats-opts.h.

4.43.1.15 OPT_NO_XLAT_CFG_NAMES

```
#define OPT_NO_XLAT_CFG_NAMES
```

Definition at line 222 of file rsstats-opts.h.

4.43.1.16 OPT_NO_XLAT_OPT_NAMES

```
#define OPT_NO_XLAT_OPT_NAMES
```

Definition at line 223 of file rsstats-opts.h.

4.43.1.17 OPT_VALUE_REPORTS

```
#define OPT_VALUE_REPORTS ((uintptr_t)DESC(REPORTS).optCookie)
```

Definition at line 152 of file rsstats-opts.h.

4.43.1.18 OPT_XLAT_CFG_NAMES

```
#define OPT_XLAT_CFG_NAMES
```

Definition at line 225 of file rsstats-opts.h.

4.43.1.19 OPT_XLAT_OPT_NAMES

```
#define OPT_XLAT_OPT_NAMES
```

Definition at line 226 of file rsstats-opts.h.

4.43.1.20 OPTION_CT

```
#define OPTION_CT 9
```

count of all options for rsstats

Definition at line 88 of file rsstats-opts.h.

4.43.1.21 REPORTS_BDBS

```
#define REPORTS_BDBS 0x1UL
```

Definition at line 149 of file rsstats-opts.h.

4.43.1.22 REPORTS_CLUSTER

```
#define REPORTS_CLUSTER 0x2UL
```

Definition at line 150 of file rsstats-opts.h.

4.43.1.23 REPORTS_MEMBERSHIP_MASK

```
#define REPORTS_MEMBERSHIP_MASK 0x3UL
```

Definition at line 151 of file rsstats-opts.h.

4.43.1.24 RESTART_OPT

```
#define RESTART_OPT(
              n )
```

Value:

```
              STMTS( \
              rsstatsOptions.curOptIdx = (n); \
              rsstatsOptions.pzCurOpt  = NULL )
```

Definition at line 173 of file rsstats-opts.h.

4.43.1.25 RSSTATS_FULL_VERSION

```
#define RSSTATS_FULL_VERSION "rsstats 0.0.1"
```

Full rsstats version text.

Definition at line 92 of file rsstats-opts.h.

4.43.1.26 RSSTATS_VERSION

```
#define RSSTATS_VERSION "0.0.1"
```

rsstats version

Definition at line 90 of file rsstats-opts.h.

4.43.1.27 SET_OPT_SAVE_OPTS

```
#define SET_OPT_SAVE_OPTS(
              a )
```

Value:

```
STMTS( \
DESC(SAVE_OPTS).fOptState &= OPTST_PERSISTENT_MASK; \
DESC(SAVE_OPTS).fOptState |= OPTST_SET; \
DESC(SAVE_OPTS).optArg.argString = (char const*)(a))
```

Definition at line 164 of file rsstats-opts.h.

4.43.1.28 STACKCT_OPT

```
#define STACKCT_OPT(
              n ) (((tArgList*)(DESC(n).optCookie))->useCt)
```

number of stacked option arguments.

Valid only for stacked option arguments.

Definition at line 119 of file rsstats-opts.h.

4.43.1.29 STACKLST_OPT

```
#define STACKLST_OPT(
              n ) (((tArgList*)(DESC(n).optCookie))->apzArgs)
```

stacked argument vector.

Valid only for stacked option arguments.

Definition at line 122 of file rsstats-opts.h.

4.43.1.30 START_OPT

```
#define START_OPT RESTART_OPT(1)
```

Definition at line 176 of file rsstats-opts.h.

4.43.1.31 STATE_OPT

```
#define STATE_OPT(
              n ) (DESC(n).fOptState & OPTST_SET_MASK)
```

Mask the option state revealing how an option was specified.

It will be one and only one of *OPTST_SET*, *OPTST_PRESET*, *OPTST_DEFINED*, *OPTST_RESET* or zero.

Definition at line 108 of file rsstats-opts.h.

4.43.1.32 USAGE

```
#define USAGE(
              c ) (*rsstatsOptions.pUsageProc)(&rsstatsOptions, c)
```

Definition at line 177 of file rsstats-opts.h.

4.43.1.33 VALUE_OPT_CLUSTERS

```
#define VALUE_OPT_CLUSTERS 'c'
```

Definition at line 146 of file rsstats-opts.h.

4.43.1.34 VALUE_OPT_HELP

```
#define VALUE_OPT_HELP 'h'
```

option flag (value) for help-value option

Definition at line 155 of file rsstats-opts.h.

4.43.1.35 VALUE_OPT_INPUT

```
#define VALUE_OPT_INPUT 'i'
```

Interface defines for specific options.

Definition at line 144 of file rsstats-opts.h.

4.43.1.36 VALUE_OPT_LOAD_OPTS

```
#define VALUE_OPT_LOAD_OPTS '<'
```

option flag (value) for load-opts-value option

Definition at line 163 of file rsstats-opts.h.

4.43.1.37 VALUE_OPT_MORE_HELP

```
#define VALUE_OPT_MORE_HELP 'H'
```

option flag (value) for more-help-value option

Definition at line 157 of file rsstats-opts.h.

4.43.1.38 VALUE_OPT_OUTPUT

```
#define VALUE_OPT_OUTPUT 'o'
```

Definition at line 145 of file rsstats-opts.h.

4.43.1.39 VALUE_OPT_REPORTS

```
#define VALUE_OPT_REPORTS 'r'
```

Definition at line 147 of file rsstats-opts.h.

4.43.1.40 VALUE_OPT_SAVE_OPTS

```
#define VALUE_OPT_SAVE_OPTS '>'
```

option flag (value) for save-opts-value option

Definition at line 161 of file rsstats-opts.h.

4.43.1.41 VALUE_OPT_VERSION

```
#define VALUE_OPT_VERSION 'v'
```

option flag (value) for version-value option

Definition at line 159 of file rsstats-opts.h.

4.43.2 Enumeration Type Documentation

4.43.2.1 rsstats_exit_code_t

```
enum rsstats_exit_code_t
```

Enumeration of rsstats exit codes.

Enumerator

RSSTATS_EXIT_SUCCESS	
RSSTATS_EXIT_FAILURE	
RSSTATS_EXIT_USAGE_ERROR	
RSSTATS_EXIT_NO_CONFIG_INPUT	
RSSTATS_EXIT_LIBOPTS_FAILURE	

Definition at line 133 of file rsstats-opts.h.

4.43.2.2 teOptIndex

```
enum teOptIndex
```

Enumeration of each option type for rsstats.

Enumerator

INDEX_OPT_INPUT	
INDEX_OPT_OUTPUT	
INDEX_OPT_CLUSTERS	
INDEX_OPT_REPORTS	
INDEX_OPT_VERSION	
INDEX_OPT_HELP	
INDEX_OPT_MORE_HELP	
INDEX_OPT_SAVE_OPTS	
INDEX_OPT_LOAD_OPTS	

Definition at line 76 of file rsstats-opts.h.

4.43.3 Variable Documentation

4.43.3.1 rsstatsOptions

```
tOptions rsstatsOptions  [extern]
```

The option definitions for rsstats.

The one structure that binds them all.

Definition at line 508 of file rsstats-opts.c.

4.44 rsstats-opts.h

Go to the documentation of this file.

```
00001 /*   -*- buffer-read-only:  t -*- vi:  set ro:
00002 *
00003 *  DO NOT EDIT THIS FILE   (rsstats-opts.h)
00004 *
00005 *  It has been AutoGen-ed
00006 *  From the definitions    rsstats-opts.def
00007 *  and the template file   options
00008 *
00009 * Generated from AutoOpts 42:1:17 templates.
00010 *
00011 *  AutoOpts is a copyrighted work.    This header file is not encumbered
00012 *  by AutoOpts licensing, but is provided under the licensing terms chosen
00013 *  by the rsstats author or copyright holder.    AutoOpts is
00014 *  licensed under the terms of the LGPL.  The redistributable library
00015 *  ("libopts") is licensed under the terms of either the LGPL or, at the
00016 *  users discretion, the BSD license.    See the AutoOpts and/or libopts sources
00017 *  for details.
00018 *
00019 * The rsstats program is copyrighted and licensed
00020 * under the following terms:
00021 *
00022 *  Copyright (C) 2024 Francois Cerbelle, all rights reserved.
00023 *  This is free software.  It is licensed for use, modification and
00024 *  redistribution under the terms of the GNU General Public License,
00025 *  version 3 or later <http://gnu.org/licenses/gpl.html>
00026 *
00027 *  rsstats is free software:  you can redistribute it and/or modify it
00028 *  under the terms of the GNU General Public License as published by the
```

```
00029 *  Free Software Foundation, either version 3 of the License, or
00030 *  (at your option) any later version.
00031 *
00032 *  rsstats is distributed in the hope that it will be useful, but
00033 *  WITHOUT ANY WARRANTY; without even the implied warranty of
00034 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
00035 *  See the GNU General Public License for more details.
00036 *
00037 *  You should have received a copy of the GNU General Public License along
00038 *  with this program.    If not, see <http://www.gnu.org/licenses/>.
00039 */
00046 #ifndef AUTOOPTS_RSSTATS_OPTS_H_GUARD
00047 #define AUTOOPTS_RSSTATS_OPTS_H_GUARD 1
00048 #include "config.h"
00049 #include <autoopts/options.h>
00050 #include <stdarg.h>
00051 #include <stdnoreturn.h>
00052
00060 #define AO_TEMPLATE_VERSION 172033
00061 #if (AO_TEMPLATE_VERSION < OPTIONS_MINIMUM_VERSION) \
00062 || (AO_TEMPLATE_VERSION > OPTIONS_STRUCT_VERSION)
00063 # error option template version mismatches autoopts/options.h header
00064   Choke Me.
00065 #endif
00066
00067 #if GCC_VERSION > 40400
00068 #define NOT_REACHED __builtin_unreachable();
00069 #else
00070 #define NOT_REACHED
00071 #endif
00072
00076 typedef enum {
00077     INDEX_OPT_INPUT       =  0,
00078     INDEX_OPT_OUTPUT      =  1,
00079     INDEX_OPT_CLUSTERS    =  2,
00080     INDEX_OPT_REPORTS     =  3,
00081     INDEX_OPT_VERSION     =  4,
00082     INDEX_OPT_HELP        =  5,
00083     INDEX_OPT_MORE_HELP   =  6,
00084     INDEX_OPT_SAVE_OPTS   =  7,
00085     INDEX_OPT_LOAD_OPTS   =  8
00086 } teOptIndex;
00088 #define OPTION_CT    9
00090 #define RSSTATS_VERSION       "0.0.1"
00092 #define RSSTATS_FULL_VERSION  "rsstats 0.0.1"
00093
00099 #define         DESC(n) (rsstatsOptions.pOptDesc[INDEX_OPT_## n])
00101 #define     HAVE_OPT(n) (!  UNUSED_OPT(& DESC(n)))
00103 #define      OPT_ARG(n) (DESC(n).optArg.argString)
00108 #define    STATE_OPT(n) (DESC(n).fOptState & OPTST_SET_MASK)
00110 #define    COUNT_OPT(n) (DESC(n).optOccCt)
00112 #define    ISSEL_OPT(n) (SELECTED_OPT(&DESC(n)))
00114 #define ISUNUSED_OPT(n) (UNUSED_OPT(& DESC(n)))
00116 #define  ENABLED_OPT(n) (!  DISABLED_OPT(& DESC(n)))
00119 #define  STACKCT_OPT(n) (((tArgList*)(DESC(n).optCookie))->useCt)
00122 #define STACKLST_OPT(n) (((tArgList*)(DESC(n).optCookie))->apzArgs)
00124 #define    CLEAR_OPT(n) STMTS( \
00125 DESC(n).fOptState &= OPTST_PERSISTENT_MASK;   \
00126 if ((DESC(n).fOptState & OPTST_INITENABLED) == 0) \
00127 DESC(n).fOptState |= OPTST_DISABLED; \
00128 DESC(n).optCookie = NULL )
00129 /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
00133 typedef enum {
00134     RSSTATS_EXIT_SUCCESS         = 0,
00135     RSSTATS_EXIT_FAILURE         = 1,
00136     RSSTATS_EXIT_USAGE_ERROR     = 64,
00137     RSSTATS_EXIT_NO_CONFIG_INPUT = 66,
00138     RSSTATS_EXIT_LIBOPTS_FAILURE = 70
00139 }   rsstats_exit_code_t;
00144 #define VALUE_OPT_INPUT          'i'
00145 #define VALUE_OPT_OUTPUT         'o'
00146 #define VALUE_OPT_CLUSTERS       'c'
00147 #define VALUE_OPT_REPORTS        'r'
00148
00149 #define REPORTS_BDBS             0x1UL
00150 #define REPORTS_CLUSTER          0x2UL
00151 #define REPORTS_MEMBERSHIP_MASK  0x3UL
00152 #define OPT_VALUE_REPORTS        ((uintptr_t)DESC(REPORTS).optCookie)
00153 #define OPT_MEMLST_REPORTS       optionMemberList(&DESC(REPORTS))
00155 #define VALUE_OPT_HELP          'h'
00157 #define VALUE_OPT_MORE_HELP     'H'
00159 #define VALUE_OPT_VERSION       'v'
00161 #define VALUE_OPT_SAVE_OPTS     '>'
00163 #define VALUE_OPT_LOAD_OPTS     '<'
00164 #define SET_OPT_SAVE_OPTS(a)   STMTS( \
00165 DESC(SAVE_OPTS).fOptState &= OPTST_PERSISTENT_MASK; \
00166 DESC(SAVE_OPTS).fOptState |= OPTST_SET; \
```

```
00167 DESC(SAVE_OPTS).optArg.argString = (char const*)(a))
00168 /*
00169 *  Interface defines not associated with particular options
00170 */
00171 #define ERRSKIP_OPTERR  STMTS(rsstatsOptions.fOptSet &= ~OPTPROC_ERRSTOP)
00172 #define ERRSTOP_OPTERR  STMTS(rsstatsOptions.fOptSet |= OPTPROC_ERRSTOP)
00173 #define RESTART_OPT(n)  STMTS( \
00174 rsstatsOptions.curOptIdx = (n); \
00175 rsstatsOptions.pzCurOpt  = NULL )
00176 #define START_OPT       RESTART_OPT(1)
00177 #define USAGE(c)        (*rsstatsOptions.pUsageProc)(&rsstatsOptions, c)
00178
00179 #ifdef  __cplusplus
00180 extern "C" {
00181 #endif
00182
00183
00184 /* * * * * *
00185 *
00186 *  Declare the rsstats option descriptor.
00187 */
00188 extern tOptions rsstatsOptions;
00189
00190 #if defined(ENABLE_NLS)
00191 # ifndef _
00192 #   include <stdio.h>
00193 #   ifndef HAVE_GETTEXT
00194       extern char * gettext(char const *);
00195 #   else
00196 #     include <libintl.h>
00197 #   endif
00198
00199 # ifndef ATTRIBUTE_FORMAT_ARG
00200 #   define ATTRIBUTE_FORMAT_ARG(_a)
00201 # endif
00202
00203 static inline char* aoGetsText(char const* pz) ATTRIBUTE_FORMAT_ARG(1);
00204 static inline char* aoGetsText(char const* pz) {
00205     if (pz == NULL) return NULL;
00206     return (char*)gettext(pz);
00207 }
00208 #   define _(s)  aoGetsText(s)
00209 # endif /* _() */
00210
00211 # define OPT_NO_XLAT_CFG_NAMES  STMTS(rsstatsOptions.fOptSet |= \
00212 OPTPROC_NXLAT_OPT_CFG;)
00213 # define OPT_NO_XLAT_OPT_NAMES  STMTS(rsstatsOptions.fOptSet |= \
00214 OPTPROC_NXLAT_OPT|OPTPROC_NXLAT_OPT_CFG;)
00215
00216 # define OPT_XLAT_CFG_NAMES     STMTS(rsstatsOptions.fOptSet &= \
00217 ~(OPTPROC_NXLAT_OPT|OPTPROC_NXLAT_OPT_CFG);)
00218 # define OPT_XLAT_OPT_NAMES     STMTS(rsstatsOptions.fOptSet &= \
00219 ~OPTPROC_NXLAT_OPT;)
00220
00221 #else   /* ENABLE_NLS */
00222 # define OPT_NO_XLAT_CFG_NAMES
00223 # define OPT_NO_XLAT_OPT_NAMES
00224
00225 # define OPT_XLAT_CFG_NAMES
00226 # define OPT_XLAT_OPT_NAMES
00227
00228 # ifndef _
00229 #   define _(_s)  _s
00230 # endif
00231 #endif  /* ENABLE_NLS */
00232
00233
00234 #ifdef  __cplusplus
00235 }
00236 #endif
00237 #endif /* AUTOOPTS_RSSTATS_OPTS_H_GUARD */
00238
00239 /* rsstats-opts.h ends here */
```

Index