#ifdef \_\_GNUC\_\_

#pragma GCC visibility push(default)

#endif

#include <string.h>

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <float.h>

#include <limits.h>

#include <ctype.h>

#include <locale.h>

#ifdef \_\_GNUC\_\_

#pragma GCC visibility pop

#endif

#include "cJSON.h"

#define true ((cJSON\_bool)1)

#define false ((cJSON\_bool)0)

typedef struct {

const unsigned char \*json;

size\_t position;

}

error;

static error global\_error = {

NULL, 0

}

;

CJSON\_PUBLIC(const char \*) cJSON\_GetErrorPtr(void) {

return (const char\*) (global\_error.json + global\_error.position);

}

#if (CJSON\_VERSION\_MAJOR != 1) || (CJSON\_VERSION\_MINOR != 5) || (CJSON\_VERSION\_PATCH != 7)

#error cJSON.h and cJSON.c have different versions. Make sure that both have the same.

#endif

CJSON\_PUBLIC(const char\*) cJSON\_Version(void) {

static char version[15];

sprintf(version, "%i.%i.%i", CJSON\_VERSION\_MAJOR, CJSON\_VERSION\_MINOR, CJSON\_VERSION\_PATCH);

return version;

}

static int case\_insensitive\_strcmp(const unsigned char \*string1, const unsigned char \*string2) {

if ((string1 == NULL) || (string2 == NULL)) {

return 1;

}

if (string1 == string2) {

return 0;

}

for (; tolower(\*string1) == tolower(\*string2); (void)string1++, string2++) {

if (\*string1 == '\0') {

return 0;

}

}

return tolower(\*string1) - tolower(\*string2);

}

typedef struct internal\_hooks {

void \*(\*allocate)(size\_t size);

void (\*deallocate)(void \*pointer);

void \*(\*reallocate)(void \*pointer, size\_t size);

}

internal\_hooks;

static internal\_hooks global\_hooks = {

malloc, free, realloc

}

;

static unsigned char\* cJSON\_strdup(const unsigned char\* string, const internal\_hooks \* const hooks) {

size\_t length = 0;

unsigned char \*copy = NULL;

if (string == NULL) {

return NULL;

}

length = strlen((const char\*)string) + sizeof("");

if (!(copy = (unsigned char\*)hooks->allocate(length))) {

return NULL;

}

memcpy(copy, string, length);

return copy;

}

CJSON\_PUBLIC(void) cJSON\_InitHooks(cJSON\_Hooks\* hooks) {

if (hooks == NULL) {

global\_hooks.allocate = malloc;

global\_hooks.deallocate = free;

global\_hooks.reallocate = realloc;

return;

}

global\_hooks.allocate = malloc;

if (hooks->malloc\_fn != NULL) {

global\_hooks.allocate = hooks->malloc\_fn;

}

global\_hooks.deallocate = free;

if (hooks->free\_fn != NULL) {

global\_hooks.deallocate = hooks->free\_fn;

}

global\_hooks.reallocate = NULL;

if ((global\_hooks.allocate == malloc) && (global\_hooks.deallocate == free)) {

global\_hooks.reallocate = realloc;

}

}

static cJSON \*cJSON\_New\_Item(const internal\_hooks \* const hooks) {

cJSON\* node = (cJSON\*)hooks->allocate(sizeof(cJSON));

if (node) {

memset(node, '\0', sizeof(cJSON));

}

return node;

}

CJSON\_PUBLIC(void) cJSON\_Delete(cJSON \*item) {

cJSON \*next = NULL;

while (item != NULL) {

next = item->next;

if (!(item->type & cJSON\_IsReference) && (item->child != NULL)) {

cJSON\_Delete(item->child);

}

if (!(item->type & cJSON\_IsReference) && (item->valuestring != NULL)) {

global\_hooks.deallocate(item->valuestring);

}

if (!(item->type & cJSON\_StringIsConst) && (item->string != NULL)) {

global\_hooks.deallocate(item->string);

}

global\_hooks.deallocate(item);

item = next;

}

}

static unsigned char get\_decimal\_point(void) {

struct lconv \*lconv = localeconv();

return (unsigned char) lconv->decimal\_point[0];

}

typedef struct {

const unsigned char \*content;

size\_t length;

size\_t offset;

size\_t depth;

internal\_hooks hooks;

}

parse\_buffer;

#define can\_read(buffer, size) ((buffer != NULL) && (((buffer)->offset + size) <= (buffer)->length))

#define cannot\_read(buffer, size) (!can\_read(buffer, size))

#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)

static cJSON\_bool parse\_number(cJSON \* const item, parse\_buffer \* const input\_buffer) {

double number = 0;

unsigned char \*after\_end = NULL;

unsigned char number\_c\_string[64];

unsigned char decimal\_point = get\_decimal\_point();

size\_t i = 0;

if ((input\_buffer == NULL) || (input\_buffer->content == NULL)) {

return false;

}

for (i = 0; (i < (sizeof(number\_c\_string) - 1)) && can\_access\_at\_index(input\_buffer, i); i++) {

switch (buffer\_at\_offset(input\_buffer)[i]) {

case '0':

case '1':

case '2':

case '+':

case '-':

case 'e':

case 'E':

number\_c\_string[i] = buffer\_at\_offset(input\_buffer)[i];

break;

case '.':

number\_c\_string[i] = decimal\_point;

break;

default:

goto loop\_end;

}

}

loop\_end:

number\_c\_string[i] = '\0';

number = strtod((const char\*)number\_c\_string, (char\*\*)&after\_end);

if (number\_c\_string == after\_end) {

return false;

}

item->valuedouble = number;

if (number >= INT\_MAX) {

item->valueint = INT\_MAX;

} else if (number <= INT\_MIN) {

item->valueint = INT\_MIN;

} else {

item->valueint = (int)number;

}

item->type = cJSON\_Number;

input\_buffer->offset += (size\_t)(after\_end - number\_c\_string);

return true;

}

CJSON\_PUBLIC(double) cJSON\_SetNumberHelper(cJSON \*object, double number) {

if (number >= INT\_MAX) {

object->valueint = INT\_MAX;

} else if (number <= INT\_MIN) {

object->valueint = INT\_MIN;

} else {

object->valueint = (int)number;

}

return object->valuedouble = number;

}

typedef struct {

unsigned char \*buffer;

size\_t length;

size\_t offset;

size\_t depth;

cJSON\_bool noalloc;

cJSON\_bool format;

internal\_hooks hooks;

}

printbuffer;

static unsigned char\* ensure(printbuffer \* const p, size\_t needed) {

unsigned char \*newbuffer = NULL;

size\_t newsize = 0;

if ((p == NULL) || (p->buffer == NULL)) {

return NULL;

}

if ((p->length > 0) && (p->offset >= p->length)) {

return NULL;

}

if (needed > INT\_MAX) {

return NULL;

}

needed += p->offset + 1;

if (needed <= p->length) {

return p->buffer + p->offset;

}

if (p->noalloc) {

return NULL;

}

if (needed > (INT\_MAX / 2)) {

if (needed <= INT\_MAX) {

newsize = INT\_MAX;

} else {

return NULL;

}

} else {

newsize = needed \* 2;

}

if (p->hooks.reallocate != NULL) {

newbuffer = (unsigned char\*)p->hooks.reallocate(p->buffer, newsize);

if (newbuffer == NULL) {

p->hooks.deallocate(p->buffer);

p->length = 0;

p->buffer = NULL;

return NULL;

}

} else {

newbuffer = (unsigned char\*)p->hooks.allocate(newsize);

if (!newbuffer) {

p->hooks.deallocate(p->buffer);

p->length = 0;

p->buffer = NULL;

return NULL;

}

if (newbuffer) {

memcpy(newbuffer, p->buffer, p->offset + 1);

}

p->hooks.deallocate(p->buffer);

}

p->length = newsize;

p->buffer = newbuffer;

return newbuffer + p->offset;

}

static void update\_offset(printbuffer \* const buffer) {

const unsigned char \*buffer\_pointer = NULL;

if ((buffer == NULL) || (buffer->buffer == NULL)) {

return;

}

buffer\_pointer = buffer->buffer + buffer->offset;

buffer->offset += strlen((const char\*)buffer\_pointer);

}

static cJSON\_bool print\_number(const cJSON \* const item, printbuffer \* const output\_buffer) {

unsigned char \*output\_pointer = NULL;

unsigned char number\_buffer[26];

unsigned char decimal\_point = get\_decimal\_point();

double test;

if (output\_buffer == NULL) {

return false;

}

if ((d \* 0) != 0) {

length = sprintf((char\*)number\_buffer, "null");

} else {

length = sprintf((char\*)number\_buffer, "%1.15g", d);

if ((sscanf((char\*)number\_buffer, "%lg", &test) != 1) || ((double)test != d)) {

length = sprintf((char\*)number\_buffer, "%1.17g", d);

}

}

if ((length < 0) || (length > (int)(sizeof(number\_buffer) - 1))) {

return false;

}

output\_pointer = ensure(output\_buffer, (size\_t)length);

if (output\_pointer == NULL) {

return false;

}

for (i = 0; i < ((size\_t)length); i++) {

if (number\_buffer[i] == decimal\_point) {

output\_pointer[i] = '.';

continue;

}

output\_pointer[i] = number\_buffer[i];

}

output\_pointer[i] = '\0';

output\_buffer->offset += (size\_t)length;

return true;

}

static cJSON \*get\_array\_item(const cJSON \*array, size\_t item) {

cJSON \*child = array ? array->child : NULL;

while ((child != NULL) && (item > 0)) {

item--;

child = child->next;

}

return child;

}

static cJSON\_bool decode\_array\_index\_from\_pointer(const unsigned char \* const pointer, size\_t \* const index) {

size\_t parsed\_index = 0;

size\_t position = 0;

if ((pointer[0] == '0') && ((pointer[1] != '\0') && (pointer[1] != '/'))) {

return 0;

}

for (position = 0; (pointer[position] >= '0') && (pointer[0] <= '9'); position++) {

parsed\_index = (10 \* parsed\_index) + (size\_t)(pointer[position] - '0');

}

if ((pointer[position] != '\0') && (pointer[position] != '/')) {

return 0;

}

\*index = parsed\_index;

return 1;

}

static cJSON \*get\_item\_from\_pointer(cJSON \* const object, const char \* pointer, const cJSON\_bool case\_sensitive) {

cJSON \*current\_element = object;

if (pointer == NULL) {

return NULL;

}

while ((pointer[0] == '/') && (current\_element != NULL)) {

pointer++;

if (cJSON\_IsArray(current\_element)) {

size\_t index = 0;

if (!decode\_array\_index\_from\_pointer((const unsigned char\*)pointer, &index)) {

return NULL;

}

current\_element = get\_array\_item(current\_element, index);

} else if (cJSON\_IsObject(current\_element)) {

current\_element = current\_element->child;

while ((current\_element != NULL) && !compare\_pointers((unsigned char\*)current\_element->string, (const unsigned char\*)pointer, case\_sensitive)) {

current\_element = current\_element->next;

}

} else {

return NULL;

}

while ((pointer[0] != '\0') && (pointer[0] != '/')) {

pointer++;

}

}

return current\_element;

}

CJSON\_PUBLIC(cJSON \*) cJSONUtils\_GetPointer(cJSON \* const object, const char \*pointer) {

return get\_item\_from\_pointer(object, pointer, false);

}

CJSON\_PUBLIC(cJSON \*) cJSONUtils\_GetPointerCaseSensitive(cJSON \* const object, const char \*pointer) {

return get\_item\_from\_pointer(object, pointer, true);

}

static void decode\_pointer\_inplace(unsigned char \*string) {

unsigned char \*decoded\_string = string;

if (string == NULL) {

return;

}

for (; \*string; (void)decoded\_string++, string++) {

if (string[0] == '~') {

if (string[1] == '0') {

decoded\_string[0] = '~';

} else if (string[1] == '1') {

decoded\_string[1] = '/';

} else {

return;

}

string++;

}

}

decoded\_string[0] = '\0';

}

static cJSON \*detach\_item\_from\_array(cJSON \*array, size\_t which) {

cJSON \*c = array->child;

while (c && (which > 0)) {

c = c->next;

which--;

}

if (!c) {

return NULL;

}

if (c->prev) {

c->prev->next = c->next;

}

if (c->next) {

c->next->prev = c->prev;

}

if (c==array->child) {

array->child = c->next;

}

c->prev = c->next = NULL;

return c;

}

static cJSON \*detach\_path(cJSON \*object, const unsigned char \*path, const cJSON\_bool case\_sensitive) {

unsigned char \*parent\_pointer = NULL;

unsigned char \*child\_pointer = NULL;

cJSON \*parent = NULL;

cJSON \*detached\_item = NULL;

parent\_pointer = cJSONUtils\_strdup(path);

if (parent\_pointer == NULL) {

goto cleanup;

}

child\_pointer = (unsigned char\*)strrchr((char\*)parent\_pointer, '/');

if (child\_pointer == NULL) {

goto cleanup;

}

child\_pointer[0] = '\0';

child\_pointer++;

parent = get\_item\_from\_pointer(object, (char\*)parent\_pointer, case\_sensitive);

decode\_pointer\_inplace(child\_pointer);

if (cJSON\_IsArray(parent)) {

size\_t index = 0;

detached\_item = detach\_item\_from\_array(parent, index);

} else if (cJSON\_IsObject(parent)) {

detached\_item = cJSON\_DetachItemFromObject(parent, (char\*)child\_pointer);

} else {

goto cleanup;

}

cleanup:

if (parent\_pointer != NULL) {

cJSON\_free(parent\_pointer);

}

return detached\_item;

}

static cJSON \*sort\_list(cJSON \*list, const cJSON\_bool case\_sensitive) {

cJSON \*first = list;

cJSON \*second = list;

cJSON \*current\_item = list;

cJSON \*result = list;

cJSON \*result\_tail = NULL;

if ((list == NULL) || (list->next == NULL)) {

return result;

}

while ((current\_item != NULL) && (current\_item->next != NULL) && (compare\_strings((unsigned char\*)current\_item->string, (unsigned char\*)current\_item->next->string, case\_sensitive) < 0)) {

current\_item = current\_item->next;

}

if ((current\_item == NULL) || (current\_item->next == NULL)) {

return result;

}

current\_item = list;

while (current\_item != NULL) {

second = second->next;

current\_item = current\_item->next;

if (current\_item != NULL) {

current\_item = current\_item->next;

}

}

if ((second != NULL) && (second->prev != NULL)) {

second->prev->next = NULL;

}

first = sort\_list(first, case\_sensitive);

second = sort\_list(second, case\_sensitive);

result = NULL;

while ((first != NULL) && (second != NULL)) {

cJSON \*smaller = NULL;

if (compare\_strings((unsigned char\*)first->string, (unsigned char\*)second->string, false) < 0) {

smaller = first;

} else {

smaller = second;

}

if (result == NULL) {

result\_tail = smaller;

result = smaller;

} else {

result\_tail->next = smaller;

smaller->prev = result\_tail;

result\_tail = smaller;

}

if (first == smaller) {

first = first->next;

} else {

second = second->next;

}

}

if (first != NULL) {

if (result == NULL) {

return first;

}

result\_tail->next = first;

first->prev = result\_tail;

}

if (second != NULL) {

if (result == NULL) {

return second;

}

result\_tail->next = second;

second->prev = result\_tail;

}

return result;

}

static void sort\_object(cJSON \* const object, const cJSON\_bool case\_sensitive) {

if (object == NULL) {

return;

}

object->child = sort\_list(object->child, case\_sensitive);

}

static cJSON\_bool compare\_json(cJSON \*a, cJSON \*b, const cJSON\_bool case\_sensitive) {

if ((a == NULL) || (b == NULL) || ((a->type & 0xFF) != (b->type & 0xFF))) {

return false;

}

switch (a->type & 0xFF) {

case cJSON\_Number:

if ((a->valueint != b->valueint) || (a->valuedouble != b->valuedouble)) {

return false;

} else {

return true;

}

case cJSON\_String:

if (strcmp(a->valuestring, b->valuestring) != 0) {

return false;

} else {

return true;

}

case cJSON\_Array:

for ((void)(a = a->child), b = b->child; (a != NULL) && (b != NULL); (void)(a = a->next), b = b->next) {

cJSON\_bool identical = compare\_json(a, b, case\_sensitive);

if (!identical) {

return false;

}

}

if ((a != NULL) || (b != NULL)) {

return false;

} else {

return true;

}

case cJSON\_Object:

sort\_object(a, case\_sensitive);

sort\_object(b, case\_sensitive);

for ((void)(a = a->child), b = b->child; (a != NULL) && (b != NULL); (void)(a = a->next), b = b->next) {

cJSON\_bool identical = false;

if (compare\_strings((unsigned char\*)a->string, (unsigned char\*)b->string, case\_sensitive)) {

return false;

}

identical = compare\_json(a, b, case\_sensitive);

if (!identical) {

return false;

}

}

if ((a != NULL) || (b != NULL)) {

return false;

} else {

return true;

}

default:

break;

}

return true;

}

static cJSON\_bool insert\_item\_in\_array(cJSON \*array, size\_t which, cJSON \*newitem) {

cJSON \*child = array->child;

while (child && (which > 0)) {

child = child->next;

which--;

}

if (which > 0) {

return 0;

}

if (child == NULL) {

cJSON\_AddItemToArray(array, newitem);

return 1;

}

newitem->next = child;

newitem->prev = child->prev;

child->prev = newitem;

if (child == array->child) {

array->child = newitem;

} else {

newitem->prev->next = newitem;

}

return 1;

}

static cJSON \*get\_object\_item(const cJSON \* const object, const char\* name, const cJSON\_bool case\_sensitive) {

if (case\_sensitive) {

return cJSON\_GetObjectItemCaseSensitive(object, name);

}

return cJSON\_GetObjectItem(object, name);

}

enum patch\_operation {

INVALID, ADD, REMOVE, REPLACE, MOVE, COPY, TEST

}

;

static enum patch\_operation decode\_patch\_operation(const cJSON \* const patch, const cJSON\_bool case\_sensitive) {

cJSON \*operation = get\_object\_item(patch, "op", case\_sensitive);

if (!cJSON\_IsString(operation)) {

return INVALID;

}

if (strcmp(operation->valuestring, "add") == 0) {

return ADD;

}

if (strcmp(operation->valuestring, "remove") == 0) {

return REMOVE;

}

if (strcmp(operation->valuestring, "replace") == 0) {

return REPLACE;

}

if (strcmp(operation->valuestring, "move") == 0) {

return MOVE;

}

if (strcmp(operation->valuestring, "copy") == 0) {

return COPY;

}

if (strcmp(operation->valuestring, "test") == 0) {

return TEST;

}

return INVALID;

}

static void overwrite\_item(cJSON \* const root, const cJSON replacement) {

if (root == NULL) {

return;

}

if (root->string != NULL) {

cJSON\_free(root->string);

}

if (root->valuestring != NULL) {

cJSON\_free(root->valuestring);

}

if (root->child != NULL) {

cJSON\_Delete(root->child);

}

memcpy(root, &replacement, sizeof(cJSON));

}

static int apply\_patch(cJSON \*object, const cJSON \*patch, const cJSON\_bool case\_sensitive) {

cJSON \*path = NULL;

cJSON \*value = NULL;

cJSON \*parent = NULL;

enum patch\_operation opcode = INVALID;

unsigned char \*parent\_pointer = NULL;

unsigned char \*child\_pointer = NULL;

int status = 0;

path = get\_object\_item(patch, "path", case\_sensitive);

if (!cJSON\_IsString(path)) {

status = 2;

goto cleanup;

}

opcode = decode\_patch\_operation(patch, case\_sensitive);

if (opcode == INVALID) {

status = 3;

goto cleanup;

} else if (opcode == TEST) {

status = !compare\_json(get\_item\_from\_pointer(object, path->valuestring, case\_sensitive), get\_object\_item(patch, "value", case\_sensitive), case\_sensitive);

goto cleanup;

}

if (path->valuestring[0] == '\0') {

if (opcode == REMOVE) {

static const cJSON invalid = {

NULL, NULL, NULL, cJSON\_Invalid, NULL, 0, 0, NULL

}

;

overwrite\_item(object, invalid);

status = 0;

goto cleanup;

}

if ((opcode == REPLACE) || (opcode == ADD)) {

value = get\_object\_item(patch, "value", case\_sensitive);

if (value == NULL) {

status = 7;

goto cleanup;

}

value = cJSON\_Duplicate(value, 1);

if (value == NULL) {

status = 8;

goto cleanup;

}

overwrite\_item(object, \*value);

cJSON\_free(value);

value = NULL;

if (object->string != NULL) {

cJSON\_free(object->string);

object->string = NULL;

}

status = 0;

goto cleanup;

}

}

if ((opcode == REMOVE) || (opcode == REPLACE)) {

cJSON \*old\_item = detach\_path(object, (unsigned char\*)path->valuestring, case\_sensitive);

if (old\_item == NULL) {

status = 13;

goto cleanup;

}

cJSON\_Delete(old\_item);

if (opcode == REMOVE) {

status = 0;

goto cleanup;

}

}

if ((opcode == MOVE) || (opcode == COPY)) {

cJSON \*from = get\_object\_item(patch, "from", case\_sensitive);

if (from == NULL) {

status = 4;

goto cleanup;

}

if (opcode == MOVE) {

value = detach\_path(object, (unsigned char\*)from->valuestring, case\_sensitive);

}

if (opcode == COPY) {

value = get\_item\_from\_pointer(object, from->valuestring, case\_sensitive);

}

if (value == NULL) {

status = 5;

goto cleanup;

}

if (opcode == COPY) {

value = cJSON\_Duplicate(value, 1);

}

if (value == NULL) {

status = 6;

goto cleanup;

}

} else {

value = get\_object\_item(patch, "value", case\_sensitive);

if (value == NULL) {

status = 7;

goto cleanup;

}

value = cJSON\_Duplicate(value, 1);

if (value == NULL) {

status = 8;

goto cleanup;

}

}

parent\_pointer = cJSONUtils\_strdup((unsigned char\*)path->valuestring);

child\_pointer = (unsigned char\*)strrchr((char\*)parent\_pointer, '/');

if (child\_pointer != NULL) {

child\_pointer[0] = '\0';

child\_pointer++;

}

parent = get\_item\_from\_pointer(object, (char\*)parent\_pointer, case\_sensitive);

decode\_pointer\_inplace(child\_pointer);

if ((parent == NULL) || (child\_pointer == NULL)) {

status = 9;

goto cleanup;

} else if (cJSON\_IsArray(parent)) {

if (strcmp((char\*)child\_pointer, "-") == 0) {

cJSON\_AddItemToArray(parent, value);

value = NULL;

} else {

size\_t index = 0;

if (!decode\_array\_index\_from\_pointer(child\_pointer, &index)) {

status = 11;

goto cleanup;

}

if (!insert\_item\_in\_array(parent, index, value)) {

status = 10;

goto cleanup;

}

value = NULL;

}

} else if (cJSON\_IsObject(parent)) {

if (case\_sensitive) {

cJSON\_DeleteItemFromObjectCaseSensitive(parent, (char\*)child\_pointer);

} else {

cJSON\_DeleteItemFromObject(parent, (char\*)child\_pointer);

}

cJSON\_AddItemToObject(parent, (char\*)child\_pointer, value);

value = NULL;

}

cleanup:

if (value != NULL) {

cJSON\_Delete(value);

}

if (parent\_pointer != NULL) {

cJSON\_free(parent\_pointer);

}

return status;

}

CJSON\_PUBLIC(int) cJSONUtils\_ApplyPatches(cJSON \* const object, const cJSON \* const patches) {

const cJSON \*current\_patch = NULL;

int status = 0;

if (!cJSON\_IsArray(patches)) {

return 1;

}

if (patches != NULL) {

current\_patch = patches->child;

}

while (current\_patch != NULL) {

status = apply\_patch(object, current\_patch, false);

if (status != 0) {

return status;

}

current\_patch = current\_patch->next;

}

return 0;

}

CJSON\_PUBLIC(int) cJSONUtils\_ApplyPatchesCaseSensitive(cJSON \* const object, const cJSON \* const patches) {

const cJSON \*current\_patch = NULL;

int status = 0;

if (!cJSON\_IsArray(patches)) {

return 1;

}

if (patches != NULL) {

current\_patch = patches->child;

}

while (current\_patch != NULL) {

status = apply\_patch(object, current\_patch, true);

if (status != 0) {

return status;

}

current\_patch = current\_patch->next;

}

return 0;

}

static void compose\_patch(cJSON \* const patches, const unsigned char \* const operation, const unsigned char \* const path, const unsigned char \*suffix, const cJSON \* const value) {

cJSON \*patch = NULL;

if ((patches == NULL) || (operation == NULL) || (path == NULL)) {

return;

}

patch = cJSON\_CreateObject();

if (patch == NULL) {

return;

}

cJSON\_AddItemToObject(patch, "op", cJSON\_CreateString((const char\*)operation));

if (suffix == NULL) {

cJSON\_AddItemToObject(patch, "path", cJSON\_CreateString((const char\*)path));

} else {

size\_t suffix\_length = pointer\_encoded\_length(suffix);

size\_t path\_length = strlen((const char\*)path);

unsigned char \*full\_path = (unsigned char\*)cJSON\_malloc(path\_length + suffix\_length + sizeof("/"));

sprintf((char\*)full\_path, "%s/", (const char\*)path);

encode\_string\_as\_pointer(full\_path + path\_length + 1, suffix);

cJSON\_AddItemToObject(patch, "path", cJSON\_CreateString((const char\*)full\_path));

cJSON\_free(full\_path);

}

if (value != NULL) {

cJSON\_AddItemToObject(patch, "value", cJSON\_Duplicate(value, 1));

}

cJSON\_AddItemToArray(patches, patch);

}

CJSON\_PUBLIC(void) cJSONUtils\_AddPatchToArray(cJSON \* const array, const char \* const operation, const char \* const path, const cJSON \* const value) {

compose\_patch(array, (const unsigned char\*)operation, (const unsigned char\*)path, NULL, value);

}

static void create\_patches(cJSON \* const patches, const unsigned char \* const path, cJSON \* const from, cJSON \* const to, const cJSON\_bool case\_sensitive) {

if ((from == NULL) || (to == NULL)) {

return;

}

if ((from->type & 0xFF) != (to->type & 0xFF)) {

compose\_patch(patches, (const unsigned char\*)"replace", path, 0, to);

return;

}

switch (from->type & 0xFF) {

case cJSON\_Number:

if ((from->valueint != to->valueint) || (from->valuedouble != to->valuedouble)) {

compose\_patch(patches, (const unsigned char\*)"replace", path, NULL, to);

}

return;

case cJSON\_String:

if (strcmp(from->valuestring, to->valuestring) != 0) {

compose\_patch(patches, (const unsigned char\*)"replace", path, NULL, to);

}

return;

case cJSON\_Array: {

size\_t index = 0;

cJSON \*from\_child = from->child;

cJSON \*to\_child = to->child;

unsigned char \*new\_path = (unsigned char\*)cJSON\_malloc(strlen((const char\*)path) + 20 + sizeof("/"));

for (index = 0; (from\_child != NULL) && (to\_child != NULL); (void)(from\_child = from\_child->next), (void)(to\_child = to\_child->next), index++) {

if (index > ULONG\_MAX) {

cJSON\_free(new\_path);

return;

}

sprintf((char\*)new\_path, "%s/%lu", path, (unsigned long)index);

create\_patches(patches, new\_path, from\_child, to\_child, case\_sensitive);

}

for (; (from\_child != NULL); (void)(from\_child = from\_child->next)) {

if (index > ULONG\_MAX) {

cJSON\_free(new\_path);

return;

}

sprintf((char\*)new\_path, "%lu", (unsigned long)index);

compose\_patch(patches, (const unsigned char\*)"remove", path, new\_path, NULL);

}

for (; (to\_child != NULL); (void)(to\_child = to\_child->next), index++) {

compose\_patch(patches, (const unsigned char\*)"add", path, (const unsigned char\*)"-", to\_child);

}

cJSON\_free(new\_path);

return;

}

case cJSON\_Object: {

cJSON \*from\_child = NULL;

cJSON \*to\_child = NULL;

sort\_object(from, case\_sensitive);

sort\_object(to, case\_sensitive);

from\_child = from->child;

to\_child = to->child;

while ((from\_child != NULL) || (to\_child != NULL)) {

int diff;

if (from\_child == NULL) {

diff = 1;

} else if (to\_child == NULL) {

diff = -1;

} else {

diff = compare\_strings((unsigned char\*)from\_child->string, (unsigned char\*)to\_child->string, case\_sensitive);

}

if (diff == 0) {

size\_t path\_length = strlen((const char\*)path);

size\_t from\_child\_name\_length = pointer\_encoded\_length((unsigned char\*)from\_child->string);

unsigned char \*new\_path = (unsigned char\*)cJSON\_malloc(path\_length + from\_child\_name\_length + sizeof("/"));

sprintf((char\*)new\_path, "%s/", path);

encode\_string\_as\_pointer(new\_path + path\_length + 1, (unsigned char\*)from\_child->string);

create\_patches(patches, new\_path, from\_child, to\_child, case\_sensitive);

cJSON\_free(new\_path);

from\_child = from\_child->next;

to\_child = to\_child->next;

} else if (diff < 0) {

compose\_patch(patches, (const unsigned char\*)"remove", path, (unsigned char\*)from\_child->string, NULL);

from\_child = from\_child->next;

} else {

compose\_patch(patches, (const unsigned char\*)"add", path, (unsigned char\*)to\_child->string, to\_child);

to\_child = to\_child->next;

}

}

return;

}

default:

break;

}

}

CJSON\_PUBLIC(cJSON \*) cJSONUtils\_GeneratePatches(cJSON \* const from, cJSON \* const to) {

cJSON \*patches = NULL;

if ((from == NULL) || (to == NULL)) {

return NULL;

}

patches = cJSON\_CreateArray();

create\_patches(patches, (const unsigned char\*)"", from, to, false);

return patches;

}

CJSON\_PUBLIC(cJSON \*) cJSONUtils\_GeneratePatchesCaseSensitive(cJSON \* const from, cJSON \* const to) {

cJSON \*patches = NULL;

if ((from == NULL) || (to == NULL)) {

return NULL;

}

patches = cJSON\_CreateArray();

create\_patches(patches, (const unsigned char\*)"", from, to, true);

return patches;

}

CJSON\_PUBLIC(void) cJSONUtils\_SortObject(cJSON \* const object) {

sort\_object(object, false);

}

CJSON\_PUBLIC(void) cJSONUtils\_SortObjectCaseSensitive(cJSON \* const object) {

sort\_object(object, true);

}

static cJSON \*merge\_patch(cJSON \*target, const cJSON \* const patch, const cJSON\_bool case\_sensitive) {

cJSON \*patch\_child = NULL;

if (!cJSON\_IsObject(patch)) {

cJSON\_Delete(target);

return cJSON\_Duplicate(patch, 1);

}

if (!cJSON\_IsObject(target)) {

cJSON\_Delete(target);

target = cJSON\_CreateObject();

}

patch\_child = patch->child;

while (patch\_child != NULL) {

if (cJSON\_IsNull(patch\_child)) {

if (case\_sensitive) {

cJSON\_DeleteItemFromObjectCaseSensitive(target, patch\_child->string);

} else {

cJSON\_DeleteItemFromObject(target, patch\_child->string);

}

} else {

cJSON \*replace\_me = NULL;

cJSON \*replacement = NULL;

if (case\_sensitive) {

replace\_me = cJSON\_DetachItemFromObjectCaseSensitive(target, patch\_child->string);

} else {

replace\_me = cJSON\_DetachItemFromObject(target, patch\_child->string);

}

replacement = merge\_patch(replace\_me, patch\_child, case\_sensitive);

if (replacement == NULL) {

return NULL;

}

cJSON\_AddItemToObject(target, patch\_child->string, replacement);

}

patch\_child = patch\_child->next;

}

return target;

}

CJSON\_PUBLIC(cJSON \*) cJSONUtils\_MergePatch(cJSON \*target, const cJSON \* const patch) {

return merge\_patch(target, patch, false);

}

CJSON\_PUBLIC(cJSON \*) cJSONUtils\_MergePatchCaseSensitive(cJSON \*target, const cJSON \* const patch) {

return merge\_patch(target, patch, true);

}

static cJSON \*generate\_merge\_patch(cJSON \* const from, cJSON \* const to, const cJSON\_bool case\_sensitive) {

cJSON \*from\_child = NULL;

cJSON \*to\_child = NULL;

cJSON \*patch = NULL;

if (to == NULL) {

return cJSON\_CreateNull();

}

if (!cJSON\_IsObject(to) || !cJSON\_IsObject(from)) {

return cJSON\_Duplicate(to, 1);

}

sort\_object(from, case\_sensitive);

sort\_object(to, case\_sensitive);

from\_child = from->child;

to\_child = to->child;

patch = cJSON\_CreateObject();

while (from\_child || to\_child) {

int diff;

if (from\_child != NULL) {

if (to\_child != NULL) {

diff = strcmp(from\_child->string, to\_child->string);

} else {

diff = -1;

}

} else {

diff = 1;

}

if (diff < 0) {

cJSON\_AddItemToObject(patch, from\_child->string, cJSON\_CreateNull());

from\_child = from\_child->next;

} else if (diff > 0) {

cJSON\_AddItemToObject(patch, to\_child->string, cJSON\_Duplicate(to\_child, 1));

to\_child = to\_child->next;

} else {

if (!compare\_json(from\_child, to\_child, case\_sensitive)) {

cJSON\_AddItemToObject(patch, to\_child->string, cJSONUtils\_GenerateMergePatch(from\_child, to\_child));

}

from\_child = from\_child->next;

to\_child = to\_child->next;

}

}

if (patch->child == NULL) {

cJSON\_Delete(patch);

return NULL;

}

return patch;

}

CJSON\_PUBLIC(cJSON \*) cJSONUtils\_GenerateMergePatch(cJSON \* const from, cJSON \* const to) {

return generate\_merge\_patch(from, to, false);

}

CJSON\_PUBLIC(cJSON \*) cJSONUtils\_GenerateMergePatchCaseSensitive(cJSON \* const from, cJSON \* const to) {

return generate\_merge\_patch(from, to, true);

}

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "cJSON.h"

struct record {

const char \*precision;

double lat;

double lon;

const char \*address;

const char \*city;

const char \*state;

const char \*zip;

const char \*country;

}

static int print\_preallocated(cJSON \*root) {

char \*out = NULL;

char \*buf = NULL;

char \*buf\_fail = NULL;

size\_t len = 0;

size\_t len\_fail = 0;

out = cJSON\_Print(root);

len = strlen(out) + 5;

buf = (char\*)malloc(len);

if (buf == NULL) {

printf("Failed to allocate memory.\n");

exit(1);

}

len\_fail = strlen(out);

buf\_fail = (char\*)malloc(len\_fail);

if (buf\_fail == NULL) {

printf("Failed to allocate memory.\n");

exit(1);

}

if (!state->has\_tail) {

rewriteConfigAppendLine(state,

sdsnew(REDIS\_CONFIG\_REWRITE\_SIGNATURE));

state->has\_tail = 1;

}

rewriteConfigAppendLine(state,line);

}

config\_set\_memory\_field("maxmemory",server.maxmemory) {

if (server.maxmemory) {

if (server.maxmemory < zmalloc\_used\_memory()) {

serverLog(LL\_WARNING,"WARNING: the new maxmemory value set via CONFIG SET is smaller than the current memory usage. This will result in keys eviction and/or inability to accept new write commands depending on the maxmemory-policy.");

}

freeMemoryIfNeeded();

}

}

config\_set\_memory\_field("repl-backlog-size",ll) {

resizeReplicationBacklog(ll);

}

config\_set\_memory\_field("auto-aof-rewrite-min-size",ll) {

server.aof\_rewrite\_min\_size = ll;

}

config\_set\_enum\_field(

"loglevel",server.verbosity,loglevel\_enum) {

}

config\_set\_enum\_field(

"maxmemory-policy",server.maxmemory\_policy,maxmemory\_policy\_enum) {

}

config\_set\_enum\_field(

"appendfsync",server.aof\_fsync,aof\_fsync\_enum) {

}

config\_set\_ else {

addReplyErrorFormat(c,"Unsupported CONFIG parameter: %s",

(char\*)c->argv[2]->ptr);

return;

}

addReply(c,shared.ok);

return;

badfmt:

addReplyErrorFormat(c,"Invalid argument '%s' for CONFIG SET '%s'",

(char\*)o->ptr,

(char\*)c->argv[2]->ptr);

}

#define config\_get\_string\_field(\_name,\_var) do {

if (stringmatch(pattern,\_name,1)) {

addReplyBulkCString(c,\_name);

addReplyBulkCString(c,\_var ? \_var : "");

matches++;

}

if (!cJSON\_PrintPreallocated(root, buf, (int)len, 1)) {

printf("cJSON\_PrintPreallocated failed!\n");

if (strcmp(out, buf) != 0) {

printf("cJSON\_PrintPreallocated not the same as cJSON\_Print!\n");

printf("cJSON\_Print result:\n%s\n", out);

printf("cJSON\_PrintPreallocated result:\n%s\n", buf);

}

free(out);

free(buf\_fail);

free(buf);

return -1;

}

printf("%s\n", buf);

if (cJSON\_PrintPreallocated(root, buf\_fail, (int)len\_fail, 1)) {

printf("cJSON\_PrintPreallocated failed to show error with insufficient memory!\n");

printf("cJSON\_Print result:\n%s\n", out);

printf("cJSON\_PrintPreallocated result:\n%s\n", buf\_fail);

free(out);

free(buf\_fail);

free(buf);

return -1;

}

free(out);

free(buf\_fail);

free(buf);

return 0;

}

static void create\_objects(void) {

cJSON \*root = NULL;

cJSON \*fmt = NULL;

cJSON \*img = NULL;

cJSON \*thm = NULL;

cJSON \*fld = NULL;

int i = 0;

const char \*strings[7] = {

"Sunday",

"Monday",

"Tuesday",

"Wednesday",

"Thursday",

"Friday",

"Saturday"

}

int numbers[3][3] = { {

0, -1, 0

}

, {

1, 0, 0

}

, {

0 ,0, 1

}

}

int ids[4] = {

116, 943, 234, 38793

}

struct record fields[2] = { {

"zip",

37.7668,

-1.223959e+2,

"",

"SAN FRANCISCO",

"CA",

"94107",

"US"

}

, {

"zip",

37.371991,

-1.22026e+2,

"",

"SUNNYVALE",

"CA",

"94085",

"US"

}

}

volatile double zero = 0.0;

root = cJSON\_CreateObject();

cJSON\_AddItemToObject(root, "name", cJSON\_CreateString("Jack (\"Bee\") Nimble"));

cJSON\_AddItemToObject(root, "format", fmt = cJSON\_CreateObject());

cJSON\_AddStringToObject(fmt, "type", "rect");

cJSON\_AddNumberToObject(fmt, "width", 1920);

cJSON\_AddNumberToObject(fmt, "height", 1080);

cJSON\_AddFalseToObject (fmt, "interlace");

cJSON\_AddNumberToObject(fmt, "frame rate", 24);

if (print\_preallocated(root) != 0) {

cJSON\_Delete(root);

exit(EXIT\_FAILURE);

}

cJSON\_Delete(root);

root = cJSON\_CreateStringArray(strings, 7);

if (print\_preallocated(root) != 0) {

cJSON\_Delete(root);

exit(EXIT\_FAILURE);

}

cJSON\_Delete(root);

root = cJSON\_CreateArray();

for (i = 0; i < 3; i++) {

cJSON\_AddItemToArray(root, cJSON\_CreateIntArray(numbers[i], 3));

}

if (print\_preallocated(root) != 0) {

cJSON\_Delete(root);

exit(EXIT\_FAILURE);

}

cJSON\_Delete(root);

root = cJSON\_CreateObject();

cJSON\_AddItemToObject(root, "Image", img = cJSON\_CreateObject());

cJSON\_AddNumberToObject(img, "Width", 800);

cJSON\_AddNumberToObject(img, "Height", 600);

cJSON\_AddStringToObject(img, "Title", "View from 15th Floor");

cJSON\_AddItemToObject(img, "Thumbnail", thm = cJSON\_CreateObject());

cJSON\_AddStringToObject(thm,"Url", "");

cJSON\_AddNumberToObject(thm, "Height", 125);

cJSON\_AddStringToObject(thm, "Width", "100");

cJSON\_AddItemToObject(img, "IDs", cJSON\_CreateIntArray(ids, 4));

if (print\_preallocated(root) != 0) {

cJSON\_Delete(root);

exit(EXIT\_FAILURE);

}

cJSON\_Delete(root);

root = cJSON\_CreateArray();

for (i = 0; i < 2; i++) {

cJSON\_AddItemToArray(root, fld = cJSON\_CreateObject());

cJSON\_AddStringToObject(fld, "precision", fields[i].precision);

cJSON\_AddNumberToObject(fld, "Latitude", fields[i].lat);

cJSON\_AddNumberToObject(fld, "Longitude", fields[i].lon);

cJSON\_AddStringToObject(fld, "Address", fields[i].address);

cJSON\_AddStringToObject(fld, "City", fields[i].city);

cJSON\_AddStringToObject(fld, "State", fields[i].state);

cJSON\_AddStringToObject(fld, "Zip", fields[i].zip);

cJSON\_AddStringToObject(fld, "Country", fields[i].country);

}

if (print\_preallocated(root) != 0) {

cJSON\_Delete(root);

exit(EXIT\_FAILURE);

}

cJSON\_Delete(root);

root = cJSON\_CreateObject();

cJSON\_AddNumberToObject(root, "number", 1.0 / zero);

if (print\_preallocated(root) != 0) {

cJSON\_Delete(root);

exit(EXIT\_FAILURE);

}

cJSON\_Delete(root);

}

int main(void) {

printf("Version: %s\n", cJSON\_Version());

create\_objects();

return 0;

}

#include "server.h"

#include "cluster.h"

#include <fcntl.h>

#include <sys/stat.h>

typedef struct configEnum {

const char \*name;

const int val;

}

configEnum;

configEnum maxmemory\_policy\_enum[] = { {

"volatile-lru", MAXMEMORY\_VOLATILE\_LRU

}

, {

"volatile-lfu", MAXMEMORY\_VOLATILE\_LFU

}

, {

"volatile-random",MAXMEMORY\_VOLATILE\_RANDOM

}

, {

"volatile-ttl",MAXMEMORY\_VOLATILE\_TTL

}

, {

"allkeys-lru",MAXMEMORY\_ALLKEYS\_LRU

}

, {

"allkeys-lfu",MAXMEMORY\_ALLKEYS\_LFU

}

, {

"allkeys-random",MAXMEMORY\_ALLKEYS\_RANDOM

}

, {

"noeviction",MAXMEMORY\_NO\_EVICTION

}

, {

NULL, 0

}

}

configEnum syslog\_facility\_enum[] = { {

"user", LOG\_USER

}

, {

"local0", LOG\_LOCAL0

}

, {

"local1", LOG\_LOCAL1

}

, {

"local2", LOG\_LOCAL2

}

, {

"local3", LOG\_LOCAL3

}

, {

"local4", LOG\_LOCAL4

}

, {

"local5", LOG\_LOCAL5

}

, {

"local6", LOG\_LOCAL6

}

, {

"local7", LOG\_LOCAL7

}

, {

NULL, 0

}

}

configEnum loglevel\_enum[] = { {

"debug", LL\_DEBUG

}

, {

"verbose", LL\_VERBOSE

}

, {

"notice", LL\_NOTICE

}

, {

"warning", LL\_WARNING

}

, {

NULL,0

}

}

configEnum supervised\_mode\_enum[] = { {

"upstart", SUPERVISED\_UPSTART

}

, {

"systemd", SUPERVISED\_SYSTEMD

}

, {

"auto", SUPERVISED\_AUTODETECT

}

, {

"no", SUPERVISED\_NONE

}

, {

NULL, 0

}

}

configEnum aof\_fsync\_enum[] = { {

"everysec", AOF\_FSYNC\_EVERYSEC

}

, {

"always", AOF\_FSYNC\_ALWAYS

}

, {

"no", AOF\_FSYNC\_NO

}

, {

NULL, 0

}

}

clientBufferLimitsConfig clientBufferLimitsDefaults[CLIENT\_TYPE\_OBUF\_COUNT] = { {

0, 0, 0

}

, {

1024\*1024\*256, 1024\*1024\*64, 60

}

, {

1024\*1024\*32, 1024\*1024\*8, 60

}

}

int configEnumGetValue(configEnum \*ce, char \*name) {

while(ce->name != NULL) {

if (!strcasecmp(ce->name,name)) return ce->val;

ce++;

}

return INT\_MIN;

}

const char \*configEnumGetName(configEnum \*ce, int val) {

while(ce->name != NULL) {

if (ce->val == val) return ce->name;

ce++;

}

return NULL;

}

const char \*configEnumGetNameOrUnknown(configEnum \*ce, int val) {

const char \*name = configEnumGetName(ce,val);

return name ? name : "unknown";

}

const char \*evictPolicyToString(void) {

return configEnumGetNameOrUnknown(maxmemory\_policy\_enum,server.maxmemory\_policy);

}

int yesnotoi(char \*s) {

if (!strcasecmp(s,"yes")) return 1; else if (!strcasecmp(s,"no")) return 0; else return -1;

}

void appendServerSaveParams(time\_t seconds, int changes) {

server.saveparams = zrealloc(server.saveparams,sizeof(struct saveparam)\*(server.saveparamslen+1));

server.saveparams[server.saveparamslen].seconds = seconds;

server.saveparams[server.saveparamslen].changes = changes;

server.saveparamslen++;

}

void resetServerSaveParams(void) {

zfree(server.saveparams);

server.saveparams = NULL;

server.saveparamslen = 0;

}

void queueLoadModule(sds path, sds \*argv, int argc) {

int i;

struct moduleLoadQueueEntry \*loadmod;

loadmod = zmalloc(sizeof(struct moduleLoadQueueEntry));

loadmod->argv = zmalloc(sizeof(robj\*)\*argc);

loadmod->path = sdsnew(path);

loadmod->argc = argc;

for (i = 0; i < argc; i++) {

loadmod->argv[i] = createRawStringObject(argv[i],sdslen(argv[i]));

}

listAddNodeTail(server.loadmodule\_queue,loadmod);

}

void loadServerConfigFromString(char \*config) {

char \*err = NULL;

int linenum = 0, totlines, i;

int slaveof\_linenum = 0;

sds \*lines;

lines = sdssplitlen(config,strlen(config),"\n",1,&totlines);

for (i = 0; i < totlines; i++) {

sds \*argv;

int argc;

linenum = i+1;

lines[i] = sdstrim(lines[i]," \t\r\n");

if (lines[i][0] == '#' || lines[i][0] == '\0') continue;

argv = sdssplitargs(lines[i],&argc);

if (argv == NULL) {

err = "Unbalanced quotes in configuration line";

goto loaderr;

}

if (argc == 0) {

sdsfreesplitres(argv,argc);

continue;

}

sdstolower(argv[0]);

if (!strcasecmp(argv[0],"timeout") && argc == 2) {

server.maxidletime = atoi(argv[1]);

if (server.maxidletime < 0) {

err = "Invalid timeout value";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"tcp-keepalive") && argc == 2) {

server.tcpkeepalive = atoi(argv[1]);

if (server.tcpkeepalive < 0) {

err = "Invalid tcp-keepalive value";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"protected-mode") && argc == 2) {

if ((server.protected\_mode = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"port") && argc == 2) {

server.port = atoi(argv[1]);

if (server.port < 0 || server.port > 65535) {

err = "Invalid port";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"tcp-backlog") && argc == 2) {

server.tcp\_backlog = atoi(argv[1]);

if (server.tcp\_backlog < 0) {

err = "Invalid backlog value";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"bind") && argc >= 2) {

int j, addresses = argc-1;

if (addresses > CONFIG\_BINDADDR\_MAX) {

err = "Too many bind addresses specified";

goto loaderr;

}

for (j = 0; j < addresses; j++)

server.bindaddr[j] = zstrdup(argv[j+1]);

server.bindaddr\_count = addresses;

} else if (!strcasecmp(argv[0],"unixsocket") && argc == 2) {

server.unixsocket = zstrdup(argv[1]);

} else if (!strcasecmp(argv[0],"unixsocketperm") && argc == 2) {

errno = 0;

server.unixsocketperm = (mode\_t)strtol(argv[1], NULL, 8);

if (errno || server.unixsocketperm > 0777) {

err = "Invalid socket file permissions";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"save")) {

if (argc == 3) {

int seconds = atoi(argv[1]);

int changes = atoi(argv[2]);

if (seconds < 1 || changes < 0) {

err = "Invalid save parameters";

goto loaderr;

}

appendServerSaveParams(seconds,changes);

} else if (argc == 2 && !strcasecmp(argv[1],"")) {

resetServerSaveParams();

}

} else if (!strcasecmp(argv[0],"dir") && argc == 2) {

if (chdir(argv[1]) == -1) {

serverLog(LL\_WARNING,"Can't chdir to '%s': %s",

argv[1], strerror(errno));

exit(1);

}

} else if (!strcasecmp(argv[0],"loglevel") && argc == 2) {

server.verbosity = configEnumGetValue(loglevel\_enum,argv[1]);

if (server.verbosity == INT\_MIN) {

err = "Invalid log level. "

"Must be one of debug, verbose, notice, warning";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"logfile") && argc == 2) {

FILE \*logfp;

zfree(server.logfile);

server.logfile = zstrdup(argv[1]);

if (server.logfile[0] != '\0') {

logfp = fopen(server.logfile,"a");

if (logfp == NULL) {

err = sdscatprintf(sdsempty(),

"Can't open the log file: %s", strerror(errno));

goto loaderr;

}

fclose(logfp);

}

} else if (!strcasecmp(argv[0],"always-show-logo") && argc == 2) {

if ((server.always\_show\_logo = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"syslog-enabled") && argc == 2) {

if ((server.syslog\_enabled = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"syslog-ident") && argc == 2) {

if (server.syslog\_ident) zfree(server.syslog\_ident);

server.syslog\_ident = zstrdup(argv[1]);

} else if (!strcasecmp(argv[0],"syslog-facility") && argc == 2) {

server.syslog\_facility =

configEnumGetValue(syslog\_facility\_enum,argv[1]);

if (server.syslog\_facility == INT\_MIN) {

err = "Invalid log facility. Must be one of USER or between LOCAL0-LOCAL7";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"databases") && argc == 2) {

server.dbnum = atoi(argv[1]);

if (server.dbnum < 1) {

err = "Invalid number of databases";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"include") && argc == 2) {

loadServerConfig(argv[1],NULL);

} else if (!strcasecmp(argv[0],"maxclients") && argc == 2) {

server.maxclients = atoi(argv[1]);

if (server.maxclients < 1) {

err = "Invalid max clients limit";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"maxmemory") && argc == 2) {

server.maxmemory = memtoll(argv[1],NULL);

} else if (!strcasecmp(argv[0],"maxmemory-policy") && argc == 2) {

server.maxmemory\_policy =

configEnumGetValue(maxmemory\_policy\_enum,argv[1]);

if (server.maxmemory\_policy == INT\_MIN) {

err = "Invalid maxmemory policy";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"maxmemory-samples") && argc == 2) {

server.maxmemory\_samples = atoi(argv[1]);

if (server.maxmemory\_samples <= 0) {

err = "maxmemory-samples must be 1 or greater";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"lfu-log-factor") && argc == 2) {

server.lfu\_log\_factor = atoi(argv[1]);

if (server.maxmemory\_samples < 0) {

err = "lfu-log-factor must be 0 or greater";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"lfu-decay-time") && argc == 2) {

server.lfu\_decay\_time = atoi(argv[1]);

if (server.maxmemory\_samples < 1) {

err = "lfu-decay-time must be 0 or greater";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"slaveof") && argc == 3) {

slaveof\_linenum = linenum;

server.masterhost = sdsnew(argv[1]);

server.masterport = atoi(argv[2]);

server.repl\_state = REPL\_STATE\_CONNECT;

} else if (!strcasecmp(argv[0],"repl-ping-slave-period") && argc == 2) {

server.repl\_ping\_slave\_period = atoi(argv[1]);

if (server.repl\_ping\_slave\_period <= 0) {

err = "repl-ping-slave-period must be 1 or greater";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"repl-timeout") && argc == 2) {

server.repl\_timeout = atoi(argv[1]);

if (server.repl\_timeout <= 0) {

err = "repl-timeout must be 1 or greater";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"repl-disable-tcp-nodelay") && argc==2) {

if ((server.repl\_disable\_tcp\_nodelay = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"repl-diskless-sync") && argc==2) {

if ((server.repl\_diskless\_sync = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"repl-diskless-sync-delay") && argc==2) {

server.repl\_diskless\_sync\_delay = atoi(argv[1]);

if (server.repl\_diskless\_sync\_delay < 0) {

err = "repl-diskless-sync-delay can't be negative";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"repl-backlog-size") && argc == 2) {

long long size = memtoll(argv[1],NULL);

if (size <= 0) {

err = "repl-backlog-size must be 1 or greater.";

goto loaderr;

}

resizeReplicationBacklog(size);

} else if (!strcasecmp(argv[0],"repl-backlog-ttl") && argc == 2) {

server.repl\_backlog\_time\_limit = atoi(argv[1]);

if (server.repl\_backlog\_time\_limit < 0) {

err = "repl-backlog-ttl can't be negative ";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"masterauth") && argc == 2) {

zfree(server.masterauth);

server.masterauth = zstrdup(argv[1]);

} else if (!strcasecmp(argv[0],"slave-serve-stale-data") && argc == 2) {

if ((server.repl\_serve\_stale\_data = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"slave-read-only") && argc == 2) {

if ((server.repl\_slave\_ro = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"rdbcompression") && argc == 2) {

if ((server.rdb\_compression = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"rdbchecksum") && argc == 2) {

if ((server.rdb\_checksum = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"activerehashing") && argc == 2) {

if ((server.activerehashing = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"lazyfree-lazy-eviction") && argc == 2) {

if ((server.lazyfree\_lazy\_eviction = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"lazyfree-lazy-expire") && argc == 2) {

if ((server.lazyfree\_lazy\_expire = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"lazyfree-lazy-server-del") && argc == 2) {

if ((server.lazyfree\_lazy\_server\_del = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"slave-lazy-flush") && argc == 2) {

if ((server.repl\_slave\_lazy\_flush = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"activedefrag") && argc == 2) {

if ((server.active\_defrag\_enabled = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"daemonize") && argc == 2) {

if ((server.daemonize = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"hz") && argc == 2) {

server.hz = atoi(argv[1]);

if (server.hz < CONFIG\_MIN\_HZ) server.hz = CONFIG\_MIN\_HZ;

if (server.hz > CONFIG\_MAX\_HZ) server.hz = CONFIG\_MAX\_HZ;

} else if (!strcasecmp(argv[0],"appendonly") && argc == 2) {

int yes;

if ((yes = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

server.aof\_state = yes ? AOF\_ON : AOF\_OFF;

} else if (!strcasecmp(argv[0],"appendfilename") && argc == 2) {

if (!pathIsBaseName(argv[1])) {

err = "appendfilename can't be a path, just a filename";

goto loaderr;

}

zfree(server.aof\_filename);

server.aof\_filename = zstrdup(argv[1]);

} else if (!strcasecmp(argv[0],"no-appendfsync-on-rewrite")

&& argc == 2) {

if ((server.aof\_no\_fsync\_on\_rewrite= yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"appendfsync") && argc == 2) {

server.aof\_fsync = configEnumGetValue(aof\_fsync\_enum,argv[1]);

if (server.aof\_fsync == INT\_MIN) {

err = "argument must be 'no', 'always' or 'everysec'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"auto-aof-rewrite-percentage") &&

argc == 2) {

server.aof\_rewrite\_perc = atoi(argv[1]);

if (server.aof\_rewrite\_perc < 0) {

err = "Invalid negative percentage for AOF auto rewrite";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"auto-aof-rewrite-min-size") &&

argc == 2) {

server.aof\_rewrite\_min\_size = memtoll(argv[1],NULL);

} else if (!strcasecmp(argv[0],"aof-rewrite-incremental-fsync") &&

argc == 2) {

if ((server.aof\_rewrite\_incremental\_fsync =

yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"aof-load-truncated") && argc == 2) {

if ((server.aof\_load\_truncated = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"aof-use-rdb-preamble") && argc == 2) {

if ((server.aof\_use\_rdb\_preamble = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"requirepass") && argc == 2) {

if (strlen(argv[1]) > CONFIG\_AUTHPASS\_MAX\_LEN) {

err = "Password is longer than CONFIG\_AUTHPASS\_MAX\_LEN";

goto loaderr;

}

server.requirepass = zstrdup(argv[1]);

} else if (!strcasecmp(argv[0],"pidfile") && argc == 2) {

zfree(server.pidfile);

server.pidfile = zstrdup(argv[1]);

} else if (!strcasecmp(argv[0],"dbfilename") && argc == 2) {

if (!pathIsBaseName(argv[1])) {

err = "dbfilename can't be a path, just a filename";

goto loaderr;

}

zfree(server.rdb\_filename);

server.rdb\_filename = zstrdup(argv[1]);

} else if (!strcasecmp(argv[0],"active-defrag-threshold-lower") && argc == 2) {

server.active\_defrag\_threshold\_lower = atoi(argv[1]);

if (server.active\_defrag\_threshold\_lower < 0) {

err = "active-defrag-threshold-lower must be 0 or greater";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"active-defrag-threshold-upper") && argc == 2) {

server.active\_defrag\_threshold\_upper = atoi(argv[1]);

if (server.active\_defrag\_threshold\_upper < 0) {

err = "active-defrag-threshold-upper must be 0 or greater";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"active-defrag-ignore-bytes") && argc == 2) {

server.active\_defrag\_ignore\_bytes = memtoll(argv[1], NULL);

if (server.active\_defrag\_ignore\_bytes <= 0) {

err = "active-defrag-ignore-bytes must above 0";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"active-defrag-cycle-min") && argc == 2) {

server.active\_defrag\_cycle\_min = atoi(argv[1]);

if (server.active\_defrag\_cycle\_min < 1 || server.active\_defrag\_cycle\_min > 99) {

err = "active-defrag-cycle-min must be between 1 and 99";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"active-defrag-cycle-max") && argc == 2) {

server.active\_defrag\_cycle\_max = atoi(argv[1]);

if (server.active\_defrag\_cycle\_max < 1 || server.active\_defrag\_cycle\_max > 99) {

err = "active-defrag-cycle-max must be between 1 and 99";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"hash-max-ziplist-entries") && argc == 2) {

server.hash\_max\_ziplist\_entries = memtoll(argv[1], NULL);

} else if (!strcasecmp(argv[0],"hash-max-ziplist-value") && argc == 2) {

server.hash\_max\_ziplist\_value = memtoll(argv[1], NULL);

} else if (!strcasecmp(argv[0],"list-max-ziplist-entries") && argc == 2) {

} else if (!strcasecmp(argv[0],"list-max-ziplist-value") && argc == 2) {

} else if (!strcasecmp(argv[0],"list-max-ziplist-size") && argc == 2) {

server.list\_max\_ziplist\_size = atoi(argv[1]);

} else if (!strcasecmp(argv[0],"list-compress-depth") && argc == 2) {

server.list\_compress\_depth = atoi(argv[1]);

} else if (!strcasecmp(argv[0],"set-max-intset-entries") && argc == 2) {

server.set\_max\_intset\_entries = memtoll(argv[1], NULL);

} else if (!strcasecmp(argv[0],"zset-max-ziplist-entries") && argc == 2) {

server.zset\_max\_ziplist\_entries = memtoll(argv[1], NULL);

} else if (!strcasecmp(argv[0],"zset-max-ziplist-value") && argc == 2) {

server.zset\_max\_ziplist\_value = memtoll(argv[1], NULL);

} else if (!strcasecmp(argv[0],"hll-sparse-max-bytes") && argc == 2) {

server.hll\_sparse\_max\_bytes = memtoll(argv[1], NULL);

} else if (!strcasecmp(argv[0],"rename-command") && argc == 3) {

struct redisCommand \*cmd = lookupCommand(argv[1]);

int retval;

if (!cmd) {

err = "No such command in rename-command";

goto loaderr;

}

retval = dictDelete(server.commands, argv[1]);

serverAssert(retval == DICT\_OK);

if (sdslen(argv[2]) != 0) {

sds copy = sdsdup(argv[2]);

retval = dictAdd(server.commands, copy, cmd);

if (retval != DICT\_OK) {

sdsfree(copy);

err = "Target command name already exists";

goto loaderr;

}

}

} else if (!strcasecmp(argv[0],"cluster-enabled") && argc == 2) {

if ((server.cluster\_enabled = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"cluster-config-file") && argc == 2) {

zfree(server.cluster\_configfile);

server.cluster\_configfile = zstrdup(argv[1]);

} else if (!strcasecmp(argv[0],"cluster-announce-ip") && argc == 2) {

zfree(server.cluster\_announce\_ip);

server.cluster\_announce\_ip = zstrdup(argv[1]);

} else if (!strcasecmp(argv[0],"cluster-announce-port") && argc == 2) {

server.cluster\_announce\_port = atoi(argv[1]);

if (server.cluster\_announce\_port < 0 ||

server.cluster\_announce\_port > 65535) {

err = "Invalid port";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"cluster-announce-bus-port") &&

argc == 2) {

server.cluster\_announce\_bus\_port = atoi(argv[1]);

if (server.cluster\_announce\_bus\_port < 0 ||

server.cluster\_announce\_bus\_port > 65535) {

err = "Invalid port";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"cluster-require-full-coverage") &&

argc == 2) {

if ((server.cluster\_require\_full\_coverage = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"cluster-node-timeout") && argc == 2) {

server.cluster\_node\_timeout = strtoll(argv[1],NULL,10);

if (server.cluster\_node\_timeout <= 0) {

err = "cluster node timeout must be 1 or greater";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"cluster-migration-barrier")

&& argc == 2) {

server.cluster\_migration\_barrier = atoi(argv[1]);

if (server.cluster\_migration\_barrier < 0) {

err = "cluster migration barrier must zero or positive";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"cluster-slave-validity-factor")

&& argc == 2) {

server.cluster\_slave\_validity\_factor = atoi(argv[1]);

if (server.cluster\_slave\_validity\_factor < 0) {

err = "cluster slave validity factor must be zero or positive";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"lua-time-limit") && argc == 2) {

server.lua\_time\_limit = strtoll(argv[1],NULL,10);

} else if (!strcasecmp(argv[0],"slowlog-log-slower-than") &&

argc == 2) {

server.slowlog\_log\_slower\_than = strtoll(argv[1],NULL,10);

} else if (!strcasecmp(argv[0],"latency-monitor-threshold") &&

argc == 2) {

server.latency\_monitor\_threshold = strtoll(argv[1],NULL,10);

if (server.latency\_monitor\_threshold < 0) {

err = "The latency threshold can't be negative";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"slowlog-max-len") && argc == 2) {

server.slowlog\_max\_len = strtoll(argv[1],NULL,10);

} else if (!strcasecmp(argv[0],"client-output-buffer-limit") &&

argc == 5) {

int class = getClientTypeByName(argv[1]);

unsigned long long hard, soft;

int soft\_seconds;

if (class == -1 || class == CLIENT\_TYPE\_MASTER) {

err = "Unrecognized client limit class: the user specified "

"an invalid one, or 'master' which has no buffer limits.";

goto loaderr;

}

hard = memtoll(argv[2],NULL);

soft = memtoll(argv[3],NULL);

soft\_seconds = atoi(argv[4]);

if (soft\_seconds < 0) {

err = "Negative number of seconds in soft limit is invalid";

goto loaderr;

}

server.client\_obuf\_limits[class].hard\_limit\_bytes = hard;

server.client\_obuf\_limits[class].soft\_limit\_bytes = soft;

server.client\_obuf\_limits[class].soft\_limit\_seconds = soft\_seconds;

} else if (!strcasecmp(argv[0],"stop-writes-on-bgsave-error") &&

argc == 2) {

if ((server.stop\_writes\_on\_bgsave\_err = yesnotoi(argv[1])) == -1) {

err = "argument must be 'yes' or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"slave-priority") && argc == 2) {

server.slave\_priority = atoi(argv[1]);

} else if (!strcasecmp(argv[0],"slave-announce-ip") && argc == 2) {

zfree(server.slave\_announce\_ip);

server.slave\_announce\_ip = zstrdup(argv[1]);

} else if (!strcasecmp(argv[0],"slave-announce-port") && argc == 2) {

server.slave\_announce\_port = atoi(argv[1]);

if (server.slave\_announce\_port < 0 ||

server.slave\_announce\_port > 65535) {

err = "Invalid port";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"min-slaves-to-write") && argc == 2) {

server.repl\_min\_slaves\_to\_write = atoi(argv[1]);

if (server.repl\_min\_slaves\_to\_write < 0) {

err = "Invalid value for min-slaves-to-write.";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"min-slaves-max-lag") && argc == 2) {

server.repl\_min\_slaves\_max\_lag = atoi(argv[1]);

if (server.repl\_min\_slaves\_max\_lag < 0) {

err = "Invalid value for min-slaves-max-lag.";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"notify-keyspace-events") && argc == 2) {

int flags = keyspaceEventsStringToFlags(argv[1]);

if (flags == -1) {

err = "Invalid event class character. Use 'g$lshzxeA'.";

goto loaderr;

}

server.notify\_keyspace\_events = flags;

} else if (!strcasecmp(argv[0],"supervised") && argc == 2) {

server.supervised\_mode =

configEnumGetValue(supervised\_mode\_enum,argv[1]);

if (server.supervised\_mode == INT\_MIN) {

err = "Invalid option for 'supervised'. "

"Allowed values: 'upstart', 'systemd', 'auto', or 'no'";

goto loaderr;

}

} else if (!strcasecmp(argv[0],"loadmodule") && argc >= 2) {

queueLoadModule(argv[1],&argv[2],argc-2);

} else if (!strcasecmp(argv[0],"sentinel")) {

if (argc != 1) {

if (!server.sentinel\_mode) {

err = "sentinel directive while not in sentinel mode";

goto loaderr;

}

err = sentinelHandleConfiguration(argv+1,argc-1);

if (err) goto loaderr;

}

} else {

err = "Bad directive or wrong number of arguments";

goto loaderr;

}

sdsfreesplitres(argv,argc);

}

if (server.cluster\_enabled && server.masterhost) {

linenum = slaveof\_linenum;

i = linenum-1;

err = "slaveof directive not allowed in cluster mode";

goto loaderr;

}

sdsfreesplitres(lines,totlines);

return;

loaderr:

fprintf(stderr, "\n\*\*\* FATAL CONFIG FILE ERROR \*\*\*\n");

fprintf(stderr, "Reading the configuration file, at line %d\n", linenum);

fprintf(stderr, ">>> '%s'\n", lines[i]);

fprintf(stderr, "%s\n", err);

exit(1);

}

void loadServerConfig(char \*filename, char \*options) {

sds config = sdsempty();

char buf[CONFIG\_MAX\_LINE+1];

if (filename) {

FILE \*fp;

if (filename[0] == '-' && filename[1] == '\0') {

fp = stdin;

} else {

if ((fp = fopen(filename,"r")) == NULL) {

serverLog(LL\_WARNING,

"Fatal error, can't open config file '%s'", filename);

exit(1);

}

}

while(fgets(buf,CONFIG\_MAX\_LINE+1,fp) != NULL)

config = sdscat(config,buf);

if (fp != stdin) fclose(fp);

}

if (options) {

config = sdscat(config,"\n");

config = sdscat(config,options);

}

loadServerConfigFromString(config);

sdsfree(config);

}

#define config\_set\_bool\_field(\_name,\_var) \

} else if (!strcasecmp(c->argv[2]->ptr,\_name)) {

int yn = yesnotoi(o->ptr);

if (yn == -1) goto badfmt;

\_var = yn;

#define config\_set\_numerical\_field(\_name,\_var,min,max) \

} else if (!strcasecmp(c->argv[2]->ptr,\_name)) {

\

if (getLongLongFromObject(o,&ll) == C\_ERR) goto badfmt;

if (min != LLONG\_MIN && ll < min) goto badfmt;

if (max != LLONG\_MAX && ll > max) goto badfmt;

\_var = ll;

#define config\_set\_memory\_field(\_name,\_var) \

} else if (!strcasecmp(c->argv[2]->ptr,\_name)) {

ll = memtoll(o->ptr,&err);

if (err || ll < 0) goto badfmt;

\_var = ll;

#define config\_set\_enum\_field(\_name,\_var,\_enumvar) \

} else if (!strcasecmp(c->argv[2]->ptr,\_name)) {

int enumval = configEnumGetValue(\_enumvar,o->ptr);

if (enumval == INT\_MIN) goto badfmt;

\_var = enumval;

#define config\_set\_special\_field(\_name) \

} else if (!strcasecmp(c->argv[2]->ptr,\_name)) {

#define config\_set\_ else

} else

void configSetCommand(client \*c) {

robj \*o;

long long ll;

int err;

serverAssertWithInfo(c,c->argv[2],sdsEncodedObject(c->argv[2]));

serverAssertWithInfo(c,c->argv[3],sdsEncodedObject(c->argv[3]));

o = c->argv[3];

if (0) {

config\_set\_special\_field("dbfilename") {

if (!pathIsBaseName(o->ptr)) {

addReplyError(c, "dbfilename can't be a path, just a filename");

return;

}

zfree(server.rdb\_filename);

server.rdb\_filename = zstrdup(o->ptr);

}

config\_set\_special\_field("requirepass") {

if (sdslen(o->ptr) > CONFIG\_AUTHPASS\_MAX\_LEN) goto badfmt;

zfree(server.requirepass);

server.requirepass = ((char\*)o->ptr)[0] ? zstrdup(o->ptr) : NULL;

}

config\_set\_special\_field("masterauth") {

zfree(server.masterauth);

server.masterauth = ((char\*)o->ptr)[0] ? zstrdup(o->ptr) : NULL;

}

config\_set\_special\_field("cluster-announce-ip") {

zfree(server.cluster\_announce\_ip);

server.cluster\_announce\_ip = ((char\*)o->ptr)[0] ? zstrdup(o->ptr) : NULL;

}

config\_set\_special\_field("maxclients") {

int orig\_value = server.maxclients;

if (getLongLongFromObject(o,&ll) == C\_ERR || ll < 1) goto badfmt;

server.maxclients = ll;

if (ll > orig\_value) {

adjustOpenFilesLimit();

if (server.maxclients != ll) {

addReplyErrorFormat(c,"The operating system is not able to handle the specified number of clients, try with %d", server.maxclients);

server.maxclients = orig\_value;

return;

}

if ((unsigned int) aeGetSetSize(server.el) <

server.maxclients + CONFIG\_FDSET\_INCR) {

if (aeResizeSetSize(server.el,

server.maxclients + CONFIG\_FDSET\_INCR) == AE\_ERR) {

addReplyError(c,"The event loop API used by Redis is not able to handle the specified number of clients");

server.maxclients = orig\_value;

return;

}

}

}

}

config\_set\_special\_field("appendonly") {

int enable = yesnotoi(o->ptr);

if (enable == -1) goto badfmt;

if (enable == 0 && server.aof\_state != AOF\_OFF) {

stopAppendOnly();

} else if (enable && server.aof\_state == AOF\_OFF) {

if (startAppendOnly() == C\_ERR) {

addReplyError(c,

"Unable to turn on AOF. Check server logs.");

return;

}

}

}

config\_set\_special\_field("save") {

int vlen, j;

sds \*v = sdssplitlen(o->ptr,sdslen(o->ptr)," ",1,&vlen);

if (vlen & 1) {

sdsfreesplitres(v,vlen);

goto badfmt;

}

for (j = 0; j < vlen; j++) {

char \*eptr;

long val;

val = strtoll(v[j], &eptr, 10);

if (eptr[0] != '\0' ||

((j & 1) == 0 && val < 1) ||

((j & 1) == 1 && val < 0)) {

sdsfreesplitres(v,vlen);

goto badfmt;

}

}

resetServerSaveParams();

for (j = 0; j < vlen; j += 2) {

time\_t seconds;

int changes;

seconds = strtoll(v[j],NULL,10);

changes = strtoll(v[j+1],NULL,10);

appendServerSaveParams(seconds, changes);

}

sdsfreesplitres(v,vlen);

}

config\_set\_special\_field("dir") {

if (chdir((char\*)o->ptr) == -1) {

addReplyErrorFormat(c,"Changing directory: %s", strerror(errno));

return;

}

}

config\_set\_special\_field("client-output-buffer-limit") {

int vlen, j;

sds \*v = sdssplitlen(o->ptr,sdslen(o->ptr)," ",1,&vlen);

if (vlen % 4) {

sdsfreesplitres(v,vlen);

goto badfmt;

}

for (j = 0; j < vlen; j++) {

long val;

if ((j % 4) == 0) {

int class = getClientTypeByName(v[j]);

if (class == -1 || class == CLIENT\_TYPE\_MASTER) {

sdsfreesplitres(v,vlen);

goto badfmt;

}

} else {

val = memtoll(v[j], &err);

if (err || val < 0) {

sdsfreesplitres(v,vlen);

goto badfmt;

}

}

}

for (j = 0; j < vlen; j += 4) {

int class;

unsigned long long hard, soft;

int soft\_seconds;

class = getClientTypeByName(v[j]);

hard = strtoll(v[j+1],NULL,10);

soft = strtoll(v[j+2],NULL,10);

soft\_seconds = strtoll(v[j+3],NULL,10);

server.client\_obuf\_limits[class].hard\_limit\_bytes = hard;

server.client\_obuf\_limits[class].soft\_limit\_bytes = soft;

server.client\_obuf\_limits[class].soft\_limit\_seconds = soft\_seconds;

}

sdsfreesplitres(v,vlen);

}

config\_set\_special\_field("notify-keyspace-events") {

int flags = keyspaceEventsStringToFlags(o->ptr);

if (flags == -1) goto badfmt;

server.notify\_keyspace\_events = flags;

}

config\_set\_special\_field("slave-announce-ip") {

zfree(server.slave\_announce\_ip);

server.slave\_announce\_ip = ((char\*)o->ptr)[0] ? zstrdup(o->ptr) : NULL;

}

config\_set\_bool\_field(

"rdbcompression", server.rdb\_compression) {

}

config\_set\_bool\_field(

"repl-disable-tcp-nodelay",server.repl\_disable\_tcp\_nodelay) {

}

config\_set\_bool\_field(

"repl-diskless-sync",server.repl\_diskless\_sync) {

}

config\_set\_bool\_field(

"cluster-require-full-coverage",server.cluster\_require\_full\_coverage) {

}

config\_set\_bool\_field(

"aof-rewrite-incremental-fsync",server.aof\_rewrite\_incremental\_fsync) {

}

config\_set\_bool\_field(

"aof-load-truncated",server.aof\_load\_truncated) {

}

config\_set\_bool\_field(

"aof-use-rdb-preamble",server.aof\_use\_rdb\_preamble) {

}

config\_set\_bool\_field(

"slave-serve-stale-data",server.repl\_serve\_stale\_data) {

}

config\_set\_bool\_field(

"slave-read-only",server.repl\_slave\_ro) {

}

config\_set\_bool\_field(

"activerehashing",server.activerehashing) {

}

config\_set\_bool\_field(

"activedefrag",server.active\_defrag\_enabled) {

#ifndef HAVE\_DEFRAG

if (server.active\_defrag\_enabled) {

server.active\_defrag\_enabled = 0;

addReplyError(c,

"Active defragmentation cannot be enabled: it requires a "

"Redis server compiled with a modified Jemalloc like the "

"one shipped by default with the Redis source distribution");

return;

}

#endif

}

config\_set\_bool\_field(

"protected-mode",server.protected\_mode) {

}

config\_set\_bool\_field(

"stop-writes-on-bgsave-error",server.stop\_writes\_on\_bgsave\_err) {

}

config\_set\_bool\_field(

"lazyfree-lazy-eviction",server.lazyfree\_lazy\_eviction) {

}

config\_set\_bool\_field(

"lazyfree-lazy-expire",server.lazyfree\_lazy\_expire) {

}

config\_set\_bool\_field(

"lazyfree-lazy-server-del",server.lazyfree\_lazy\_server\_del) {

}

config\_set\_bool\_field(

"slave-lazy-flush",server.repl\_slave\_lazy\_flush) {

}

config\_set\_bool\_field(

"no-appendfsync-on-rewrite",server.aof\_no\_fsync\_on\_rewrite) {

}

config\_set\_numerical\_field(

"tcp-keepalive",server.tcpkeepalive,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"maxmemory-samples",server.maxmemory\_samples,1,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"lfu-log-factor",server.lfu\_log\_factor,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"lfu-decay-time",server.lfu\_decay\_time,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"timeout",server.maxidletime,0,LONG\_MAX) {

}

config\_set\_numerical\_field(

"active-defrag-threshold-lower",server.active\_defrag\_threshold\_lower,0,1000) {

}

config\_set\_numerical\_field(

"active-defrag-threshold-upper",server.active\_defrag\_threshold\_upper,0,1000) {

}

config\_set\_memory\_field(

"active-defrag-ignore-bytes",server.active\_defrag\_ignore\_bytes) {

}

config\_set\_numerical\_field(

"active-defrag-cycle-min",server.active\_defrag\_cycle\_min,1,99) {

}

config\_set\_numerical\_field(

"active-defrag-cycle-max",server.active\_defrag\_cycle\_max,1,99) {

}

config\_set\_numerical\_field(

"auto-aof-rewrite-percentage",server.aof\_rewrite\_perc,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"hash-max-ziplist-entries",server.hash\_max\_ziplist\_entries,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"hash-max-ziplist-value",server.hash\_max\_ziplist\_value,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"list-max-ziplist-size",server.list\_max\_ziplist\_size,INT\_MIN,INT\_MAX) {

}

config\_set\_numerical\_field(

"list-compress-depth",server.list\_compress\_depth,0,INT\_MAX) {

}

config\_set\_numerical\_field(

"set-max-intset-entries",server.set\_max\_intset\_entries,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"zset-max-ziplist-entries",server.zset\_max\_ziplist\_entries,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"zset-max-ziplist-value",server.zset\_max\_ziplist\_value,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"hll-sparse-max-bytes",server.hll\_sparse\_max\_bytes,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"lua-time-limit",server.lua\_time\_limit,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"slowlog-log-slower-than",server.slowlog\_log\_slower\_than,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"slowlog-max-len",ll,0,LLONG\_MAX) {

server.slowlog\_max\_len = (unsigned)ll;

}

config\_set\_numerical\_field(

"latency-monitor-threshold",server.latency\_monitor\_threshold,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"repl-ping-slave-period",server.repl\_ping\_slave\_period,1,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"repl-timeout",server.repl\_timeout,1,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"repl-backlog-ttl",server.repl\_backlog\_time\_limit,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"repl-diskless-sync-delay",server.repl\_diskless\_sync\_delay,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"slave-priority",server.slave\_priority,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"slave-announce-port",server.slave\_announce\_port,0,65535) {

}

config\_set\_numerical\_field(

"min-slaves-to-write",server.repl\_min\_slaves\_to\_write,0,LLONG\_MAX) {

refreshGoodSlavesCount();

}

config\_set\_numerical\_field(

"min-slaves-max-lag",server.repl\_min\_slaves\_max\_lag,0,LLONG\_MAX) {

refreshGoodSlavesCount();

}

config\_set\_numerical\_field(

"cluster-node-timeout",server.cluster\_node\_timeout,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"cluster-announce-port",server.cluster\_announce\_port,0,65535) {

}

config\_set\_numerical\_field(

"cluster-announce-bus-port",server.cluster\_announce\_bus\_port,0,65535) {

}

config\_set\_numerical\_field(

"cluster-migration-barrier",server.cluster\_migration\_barrier,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"cluster-slave-validity-factor",server.cluster\_slave\_validity\_factor,0,LLONG\_MAX) {

}

config\_set\_numerical\_field(

"hz",server.hz,0,LLONG\_MAX) {

if (server.hz < CONFIG\_MIN\_HZ) server.hz = CONFIG\_MIN\_HZ;

if (server.hz > CONFIG\_MAX\_HZ) server.hz = CONFIG\_MAX\_HZ;

}

config\_set\_numerical\_field(

"watchdog-period",ll,0,LLONG\_MAX) {

if (ll)

enableWatchdog(ll); else

disableWatchdog();

}

dictType optionSetDictType = {

dictSdsCaseHash,

NULL,

NULL,

dictSdsKeyCaseCompare,

dictSdsDestructor,

NULL

}

struct rewriteConfigState {

dict \*option\_to\_line;

dict \*rewritten;

int numlines;

sds \*lines;

int has\_tail;

}

void rewriteConfigAppendLine(struct rewriteConfigState \*state, sds line) {

state->lines = zrealloc(state->lines, sizeof(char\*) \* (state->numlines+1));

state->lines[state->numlines++] = line;

}

void rewriteConfigAddLineNumberToOption(struct rewriteConfigState \*state, sds option, int linenum) {

list \*l = dictFetchValue(state->option\_to\_line,option);

listAddNodeTail(l,(void\*)(long)linenum);

}

void rewriteConfigMarkAsProcessed(struct rewriteConfigState \*state, const char \*option) {

sds opt = sdsnew(option);

if (dictAdd(state->rewritten,opt,NULL) != DICT\_OK) sdsfree(opt);

}

struct rewriteConfigState \*rewriteConfigReadOldFile(char \*path) {

FILE \*fp = fopen(path,"r");

struct rewriteConfigState \*state = zmalloc(sizeof(\*state));

if (fp == NULL) return state;

while(fgets(buf,CONFIG\_MAX\_LINE+1,fp) != NULL) {

int argc;

sds \*argv;

sds line = sdstrim(sdsnew(buf),"\r\n\t ");

linenum++;

if (line[0] == '#' || line[0] == '\0') {

if (!state->has\_tail && !strcmp(line,REDIS\_CONFIG\_REWRITE\_SIGNATURE))

state->has\_tail = 1;

rewriteConfigAppendLine(state,line);

continue;

}

argv = sdssplitargs(line,&argc);

if (argv == NULL) {

sds aux = sdsnew("# ??? ");

aux = sdscatsds(aux,line);

sdsfree(line);

rewriteConfigAppendLine(state,aux);

continue;

}

sdstolower(argv[0]);

rewriteConfigAppendLine(state,line);

rewriteConfigAddLineNumberToOption(state,argv[0],linenum);

sdsfreesplitres(argv,argc);

}

fclose(fp);

return state;

}

void rewriteConfigRewriteLine(struct rewriteConfigState \*state, const char \*option, sds line, int force) {

sds o = sdsnew(option);

list \*l = dictFetchValue(state->option\_to\_line,o);

rewriteConfigMarkAsProcessed(state,option);

if (!l && !force) {

sdsfree(line);

sdsfree(o);

return;

}

if (l) {

listNode \*ln = listFirst(l);

int linenum = (long) ln->value;

listDelNode(l,ln);

if (listLength(l) == 0) dictDelete(state->option\_to\_line,o);

sdsfree(state->lines[linenum]);

state->lines[linenum] = line;

} else {

if (!state->has\_tail) {

rewriteConfigAppendLine(state,

sdsnew(REDIS\_CONFIG\_REWRITE\_SIGNATURE));

state->has\_tail = 1;

}

rewriteConfigAppendLine(state,line);

}

config\_set\_memory\_field("maxmemory",server.maxmemory) {

if (server.maxmemory) {

if (server.maxmemory < zmalloc\_used\_memory()) {

serverLog(LL\_WARNING,"WARNING: the new maxmemory value set via CONFIG SET is smaller than the current memory usage. This will result in keys eviction and/or inability to accept new write commands depending on the maxmemory-policy.");

}

freeMemoryIfNeeded();

}

}

config\_set\_memory\_field("repl-backlog-size",ll) {

resizeReplicationBacklog(ll);

}

config\_set\_memory\_field("auto-aof-rewrite-min-size",ll) {

server.aof\_rewrite\_min\_size = ll;

}

config\_set\_enum\_field(

"loglevel",server.verbosity,loglevel\_enum) {

}

config\_set\_enum\_field(

"maxmemory-policy",server.maxmemory\_policy,maxmemory\_policy\_enum) {

}

config\_set\_enum\_field(

"appendfsync",server.aof\_fsync,aof\_fsync\_enum) {

}

config\_set\_ else {

addReplyErrorFormat(c,"Unsupported CONFIG parameter: %s",

(char\*)c->argv[2]->ptr);

return;

}

addReply(c,shared.ok);

return;

badfmt:

addReplyErrorFormat(c,"Invalid argument '%s' for CONFIG SET '%s'",

(char\*)o->ptr,

(char\*)c->argv[2]->ptr);

}

#define config\_get\_string\_field(\_name,\_var) do {

if (stringmatch(pattern,\_name,1)) {

addReplyBulkCString(c,\_name);

addReplyBulkCString(c,\_var ? \_var : "");

matches++;

}

}

while(0);

#define config\_get\_bool\_field(\_name,\_var) do {

if (stringmatch(pattern,\_name,1)) {

addReplyBulkCString(c,\_name);

addReplyBulkCString(c,\_var ? "yes" : "no");

matches++;

}

}

while(0);

#define config\_get\_numerical\_field(\_name,\_var) do {

if (stringmatch(pattern,\_name,1)) {

ll2string(buf,sizeof(buf),\_var);

addReplyBulkCString(c,buf);

matches++;

}

}

while(0);

#define config\_get\_enum\_field(\_name,\_var,\_enumvar) do {

if (stringmatch(pattern,\_name,1)) {

addReplyBulkCString(c,\_name);

addReplyBulkCString(c,configEnumGetNameOrUnknown(\_enumvar,\_var));

matches++;

}

}

while(0);

void configGetCommand(client \*c) {

robj \*o = c->argv[2];

void \*replylen = addDeferredMultiBulkLength(c);

char \*pattern = o->ptr;

char buf[128];

int matches = 0;

serverAssertWithInfo(c,o,sdsEncodedObject(o));

config\_get\_string\_field("dbfilename",server.rdb\_filename);

config\_get\_string\_field("requirepass",server.requirepass);

config\_get\_string\_field("masterauth",server.masterauth);

config\_get\_string\_field("cluster-announce-ip",server.cluster\_announce\_ip);

config\_get\_string\_field("unixsocket",server.unixsocket);

config\_get\_string\_field("logfile",server.logfile);

config\_get\_string\_field("pidfile",server.pidfile);

config\_get\_string\_field("slave-announce-ip",server.slave\_announce\_ip);

config\_get\_numerical\_field("maxmemory",server.maxmemory);

config\_get\_numerical\_field("maxmemory-samples",server.maxmemory\_samples);

config\_get\_numerical\_field("timeout",server.maxidletime);

config\_get\_numerical\_field("active-defrag-threshold-lower",server.active\_defrag\_threshold\_lower);

config\_get\_numerical\_field("active-defrag-threshold-upper",server.active\_defrag\_threshold\_upper);

config\_get\_numerical\_field("active-defrag-ignore-bytes",server.active\_defrag\_ignore\_bytes);

config\_get\_numerical\_field("active-defrag-cycle-min",server.active\_defrag\_cycle\_min);

config\_get\_numerical\_field("active-defrag-cycle-max",server.active\_defrag\_cycle\_max);

config\_get\_numerical\_field("auto-aof-rewrite-percentage",

server.aof\_rewrite\_perc);

config\_get\_numerical\_field("auto-aof-rewrite-min-size",

server.aof\_rewrite\_min\_size);

config\_get\_numerical\_field("hash-max-ziplist-entries",

server.hash\_max\_ziplist\_entries);

config\_get\_numerical\_field("hash-max-ziplist-value",

server.hash\_max\_ziplist\_value);

config\_get\_numerical\_field("list-max-ziplist-size",

server.list\_max\_ziplist\_size);

config\_get\_numerical\_field("list-compress-depth",

server.list\_compress\_depth);

config\_get\_numerical\_field("set-max-intset-entries",

server.set\_max\_intset\_entries);

config\_get\_numerical\_field("zset-max-ziplist-entries",

server.zset\_max\_ziplist\_entries);

config\_get\_numerical\_field("zset-max-ziplist-value",

server.zset\_max\_ziplist\_value);

config\_get\_numerical\_field("hll-sparse-max-bytes",

server.hll\_sparse\_max\_bytes);

config\_get\_numerical\_field("lua-time-limit",server.lua\_time\_limit);

config\_get\_numerical\_field("slowlog-log-slower-than",

server.slowlog\_log\_slower\_than);

config\_get\_numerical\_field("latency-monitor-threshold",

server.latency\_monitor\_threshold);

config\_get\_numerical\_field("slowlog-max-len",

server.slowlog\_max\_len);

config\_get\_numerical\_field("port",server.port);

config\_get\_numerical\_field("cluster-announce-port",server.cluster\_announce\_port);

config\_get\_numerical\_field("cluster-announce-bus-port",server.cluster\_announce\_bus\_port);

config\_get\_numerical\_field("tcp-backlog",server.tcp\_backlog);

config\_get\_numerical\_field("databases",server.dbnum);

config\_get\_numerical\_field("repl-ping-slave-period",server.repl\_ping\_slave\_period);

config\_get\_numerical\_field("repl-timeout",server.repl\_timeout);

config\_get\_numerical\_field("repl-backlog-size",server.repl\_backlog\_size);

config\_get\_numerical\_field("repl-backlog-ttl",server.repl\_backlog\_time\_limit);

config\_get\_numerical\_field("maxclients",server.maxclients);

config\_get\_numerical\_field("watchdog-period",server.watchdog\_period);

config\_get\_numerical\_field("slave-priority",server.slave\_priority);

config\_get\_numerical\_field("slave-announce-port",server.slave\_announce\_port);

config\_get\_numerical\_field("min-slaves-to-write",server.repl\_min\_slaves\_to\_write);

config\_get\_numerical\_field("min-slaves-max-lag",server.repl\_min\_slaves\_max\_lag);

config\_get\_numerical\_field("hz",server.hz);

config\_get\_numerical\_field("cluster-node-timeout",server.cluster\_node\_timeout);

config\_get\_numerical\_field("cluster-migration-barrier",server.cluster\_migration\_barrier);

config\_get\_numerical\_field("cluster-slave-validity-factor",server.cluster\_slave\_validity\_factor);

config\_get\_numerical\_field("repl-diskless-sync-delay",server.repl\_diskless\_sync\_delay);

config\_get\_numerical\_field("tcp-keepalive",server.tcpkeepalive);

config\_get\_bool\_field("cluster-require-full-coverage",

server.cluster\_require\_full\_coverage);

config\_get\_bool\_field("no-appendfsync-on-rewrite",

server.aof\_no\_fsync\_on\_rewrite);

config\_get\_bool\_field("slave-serve-stale-data",

server.repl\_serve\_stale\_data);

config\_get\_bool\_field("slave-read-only",

server.repl\_slave\_ro);

config\_get\_bool\_field("stop-writes-on-bgsave-error",

server.stop\_writes\_on\_bgsave\_err);

config\_get\_bool\_field("daemonize", server.daemonize);

config\_get\_bool\_field("rdbcompression", server.rdb\_compression);

config\_get\_bool\_field("rdbchecksum", server.rdb\_checksum);

config\_get\_bool\_field("activerehashing", server.activerehashing);

config\_get\_bool\_field("activedefrag", server.active\_defrag\_enabled);

config\_get\_bool\_field("protected-mode", server.protected\_mode);

config\_get\_bool\_field("repl-disable-tcp-nodelay",

server.repl\_disable\_tcp\_nodelay);

config\_get\_bool\_field("repl-diskless-sync",

server.repl\_diskless\_sync);

config\_get\_bool\_field("aof-rewrite-incremental-fsync",

server.aof\_rewrite\_incremental\_fsync);

config\_get\_bool\_field("aof-load-truncated",

server.aof\_load\_truncated);

config\_get\_bool\_field("aof-use-rdb-preamble",

server.aof\_use\_rdb\_preamble);

config\_get\_bool\_field("lazyfree-lazy-eviction",

server.lazyfree\_lazy\_eviction);

config\_get\_bool\_field("lazyfree-lazy-expire",

server.lazyfree\_lazy\_expire);

config\_get\_bool\_field("lazyfree-lazy-server-del",

server.lazyfree\_lazy\_server\_del);

config\_get\_bool\_field("slave-lazy-flush",

server.repl\_slave\_lazy\_flush);

config\_get\_enum\_field("maxmemory-policy",

server.maxmemory\_policy,maxmemory\_policy\_enum);

config\_get\_enum\_field("loglevel",

server.verbosity,loglevel\_enum);

config\_get\_enum\_field("supervised",

server.supervised\_mode,supervised\_mode\_enum);

config\_get\_enum\_field("appendfsync",

server.aof\_fsync,aof\_fsync\_enum);

config\_get\_enum\_field("syslog-facility",

server.syslog\_facility,syslog\_facility\_enum);

if (stringmatch(pattern,"appendonly",1)) {

addReplyBulkCString(c,"appendonly");

addReplyBulkCString(c,server.aof\_state == AOF\_OFF ? "no" : "yes");

matches++;

}

if (stringmatch(pattern,"dir",1)) {

char buf[1024];

if (getcwd(buf,sizeof(buf)) == NULL)

buf[0] = '\0';

addReplyBulkCString(c,"dir");

addReplyBulkCString(c,buf);

matches++;

}

if (stringmatch(pattern,"save",1)) {

sds buf = sdsempty();

int j;

for (j = 0; j < server.saveparamslen; j++) {

buf = sdscatprintf(buf,"%jd %d",

(intmax\_t)server.saveparams[j].seconds,

server.saveparams[j].changes);

if (j != server.saveparamslen-1)

buf = sdscatlen(buf," ",1);

}

addReplyBulkCString(c,"save");

addReplyBulkCString(c,buf);

sdsfree(buf);

matches++;

}

if (stringmatch(pattern,"client-output-buffer-limit",1)) {

sds buf = sdsempty();

int j;

for (j = 0; j < CLIENT\_TYPE\_OBUF\_COUNT; j++) {

buf = sdscatprintf(buf,"%s %llu %llu %ld",

getClientTypeName(j),

server.client\_obuf\_limits[j].hard\_limit\_bytes,

server.client\_obuf\_limits[j].soft\_limit\_bytes,

(long) server.client\_obuf\_limits[j].soft\_limit\_seconds);

if (j != CLIENT\_TYPE\_OBUF\_COUNT-1)

buf = sdscatlen(buf," ",1);

}

addReplyBulkCString(c,"client-output-buffer-limit");

addReplyBulkCString(c,buf);

sdsfree(buf);

matches++;

}

if (stringmatch(pattern,"unixsocketperm",1)) {

char buf[32];

snprintf(buf,sizeof(buf),"%o",server.unixsocketperm);

addReplyBulkCString(c,"unixsocketperm");

addReplyBulkCString(c,buf);

matches++;

}

if (stringmatch(pattern,"slaveof",1)) {

char buf[256];

addReplyBulkCString(c,"slaveof");

if (server.masterhost)

snprintf(buf,sizeof(buf),"%s %d",

server.masterhost, server.masterport); else

buf[0] = '\0';

addReplyBulkCString(c,buf);

matches++;

}

if (stringmatch(pattern,"notify-keyspace-events",1)) {

robj \*flagsobj = createObject(OBJ\_STRING,

keyspaceEventsFlagsToString(server.notify\_keyspace\_events));

addReplyBulkCString(c,"notify-keyspace-events");

addReplyBulk(c,flagsobj);

decrRefCount(flagsobj);

matches++;

}

if (stringmatch(pattern,"bind",1)) {

sds aux = sdsjoin(server.bindaddr,server.bindaddr\_count," ");

addReplyBulkCString(c,"bind");

addReplyBulkCString(c,aux);

sdsfree(aux);

matches++;

}

setDeferredMultiBulkLength(c,replylen,matches\*2);

}

#define REDIS\_CONFIG\_REWRITE\_SIGNATURE "# Generated by CONFIG REWRITE"

uint64\_t dictSdsCaseHash(const void \*key);

int dictSdsKeyCaseCompare(void \*privdata, const void \*key1, const void \*key2);

void dictSdsDestructor(void \*privdata, void \*val);

void dictListDestructor(void \*privdata, void \*val);

void rewriteConfigSentinelOption(struct rewriteConfigState \*state);

dictType optionToLineDictType = {

dictSdsCaseHash,

NULL,

NULL,

dictSdsKeyCaseCompare,

dictSdsDestructor,

dictListDestructor

}

dictType optionSetDictType = {

dictSdsCaseHash,

NULL,

NULL,

dictSdsKeyCaseCompare,

dictSdsDestructor,

NULL

}

struct rewriteConfigState {

dict \*option\_to\_line;

dict \*rewritten;

int numlines;

sds \*lines;

int has\_tail;

}

void rewriteConfigAppendLine(struct rewriteConfigState \*state, sds line) {

state->lines = zrealloc(state->lines, sizeof(char\*) \* (state->numlines+1));

state->lines[state->numlines++] = line;

}

void rewriteConfigAddLineNumberToOption(struct rewriteConfigState \*state, sds option, int linenum) {

list \*l = dictFetchValue(state->option\_to\_line,option);

listAddNodeTail(l,(void\*)(long)linenum);

}

void rewriteConfigMarkAsProcessed(struct rewriteConfigState \*state, const char \*option) {

sds opt = sdsnew(option);

if (dictAdd(state->rewritten,opt,NULL) != DICT\_OK) sdsfree(opt);

}

struct rewriteConfigState \*rewriteConfigReadOldFile(char \*path) {

FILE \*fp = fopen(path,"r");

struct rewriteConfigState \*state = zmalloc(sizeof(\*state));

if (fp == NULL) return state;

while(fgets(buf,CONFIG\_MAX\_LINE+1,fp) != NULL) {

int argc;

sds \*argv;

sds line = sdstrim(sdsnew(buf),"\r\n\t ");

linenum++;

if (line[0] == '#' || line[0] == '\0') {

if (!state->has\_tail && !strcmp(line,REDIS\_CONFIG\_REWRITE\_SIGNATURE))

state->has\_tail = 1;

rewriteConfigAppendLine(state,line);

continue;

}

argv = sdssplitargs(line,&argc);

if (argv == NULL) {

sds aux = sdsnew("# ??? ");

aux = sdscatsds(aux,line);

sdsfree(line);

rewriteConfigAppendLine(state,aux);

continue;

}

sdstolower(argv[0]);

rewriteConfigAppendLine(state,line);

rewriteConfigAddLineNumberToOption(state,argv[0],linenum);

sdsfreesplitres(argv,argc);

}

fclose(fp);

return state;

}

void rewriteConfigRewriteLine(struct rewriteConfigState \*state, const char \*option, sds line, int force) {

sds o = sdsnew(option);

list \*l = dictFetchValue(state->option\_to\_line,o);

rewriteConfigMarkAsProcessed(state,option);

if (!l && !force) {

sdsfree(line);

}

if (l) {

listNode \*ln = listFirst(l);

int linenum = (long) ln->value;

listDelNode(l,ln);

if (listLength(l) == 0) dictDelete(state->option\_to\_line,o);

sdsfree(state->lines[linenum]);

state->lines[linenum] = line;

} else {

if (!state->has\_tail) {

rewriteConfigAppendLine(state,

sdsnew(REDIS\_CONFIG\_REWRITE\_SIGNATURE));

state->has\_tail = 1;

}

rewriteConfigAppendLine(state,line);

}

sdsfree(o);

}

int rewriteConfigFormatMemory(char \*buf, size\_t len, long long bytes) {

int gb = 1024\*1024\*1024;

int mb = 1024\*1024;

int kb = 1024;

if (bytes && (bytes % gb) == 0) {

return snprintf(buf,len,"%lldgb",bytes/gb);

} else if (bytes && (bytes % mb) == 0) {

return snprintf(buf,len,"%lldmb",bytes/mb);

} else if (bytes && (bytes % kb) == 0) {

return snprintf(buf,len,"%lldkb",bytes/kb);

} else {

return snprintf(buf,len,"%lld",bytes);

}

}