Wypożyczalnia samochodów

Generated on Sun Jun 11 2023 00:08:18 for Wypożyczalnia samochodów by Doxygen 1.9.7

Sun Jun 11 2023 00:08:18

Chapter 1

Todo List

Member clientsMenu (void)

implement submenus.

Member formHandleIteraction (FORM ∗form)

Resign form form, change return type.

Display information about invalid data in current field.

File list.h

move function documentation into .c file.

Member menuInvoke (const char ∗const title, const char ∗const choices[], const int choicesCount, void(∗menuFun[ ])(void))

Split into functions, make allocation and dallcation seperate functions, make it allocate on heap instead of stack.

Member rentalsMenu (void)

implemnet submenus

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

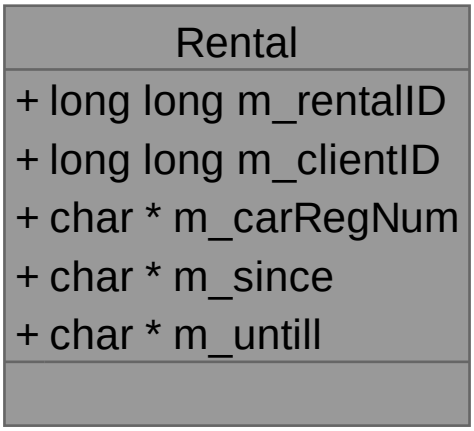
Class Documentation

4.1 Rental Struct Reference

Contains infomation about rental.

`#include <rental.h>`

Collaboration diagram for Rental:



Public Attributes

- long long m_rentalID
  Rental Id.
- long long m_clientID
  Card id of client who rents car.
- char * m_carRegNum
  Car registraction number of rented car.
- char * m_since
  Rental start date.
- char * m_untill
  Rental end date.

4.1.1 Detailed Description

Contains infomation about rental.

Definition at line 6 of file rental.h.

4.1.2 Member Data Documentation

4.1.2.1 m_carRegNum

```
char* Rental::m_carRegNum
```

Car registraction number of rented car.

Definition at line 12 of file rental.h.

4.1.2.2 m_clientID

```
long long Rental::m_clientID
```

Card id of client who rents car.

Definition at line 10 of file rental.h.

4.1.2.3 m_rentalID

```
long long Rental::m_rentalID
```

Rental Id.

Definition at line 8 of file rental.h.

4.1.2.4 m_since

```
char* Rental::m_since
```

Rental start date.

Definition at line 14 of file rental.h.

4.1.2.5 m_untill

```
char* Rental::m_untill
```

Rental end date.

Definition at line 16 of file rental.h.

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

- rental.h

Chapter 5

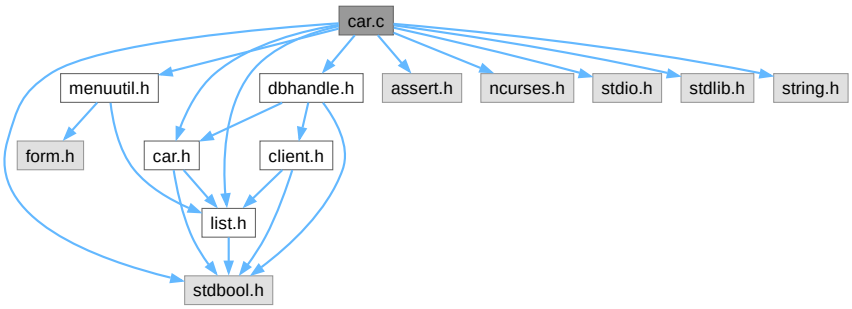
File Documentation

5.1 car.c File Reference

Implements function to operate on client.

```
#include ¨car.h¨
#include ¨dbhandle.h¨
#include ¨list.h¨
#include ¨menuutil.h¨
#include <assert.h>
#include <ncurses.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for car.c:



Functions

- struct Car ∗ carNew ()

  Allocates and returns car containing nothing.
- void carFree (struct Car ∗car)

  Deallocates car.
- bool carIsComplete (const struct Car ∗car)

Checks if every field in Car is set.

- static int carGetListQueryCallback (struct List *list, int argc, const char **argv, const char **const colNames)

  CallBack function for dbHandleGetResultAsList. Transforms row into Car.
- char * carGetQueryOfSort (int sType, bool desc)

  Generates SQL query.
- struct List * carGetList (int sType, bool desc)

  Get list of cars.
- void carClone (struct Car **dest, const struct Car *src)

  Make a clone of Car. It allocates memory internally.

5.1.1 Detailed Description

Implements function to operate on client.

Definition in file car.c.

5.1.2 Function Documentation

5.1.2.1 carClone()

```
void carClone (
            struct Car ** dest,
            const struct Car * src )
```

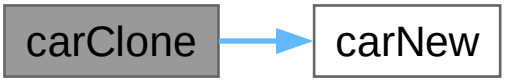
Make a clone of Car. It allocates memory internally.

Parameters

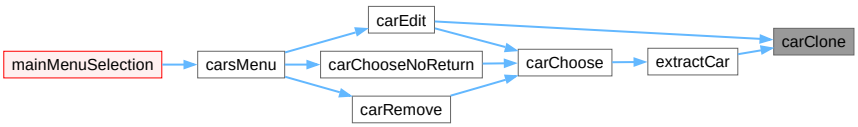
dest	Car structure where data will be cloned into.
src	Car to create a clone of.

Definition at line 168 of file car.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.2 carFree()

```
void carFree (
            struct Car * car )
```

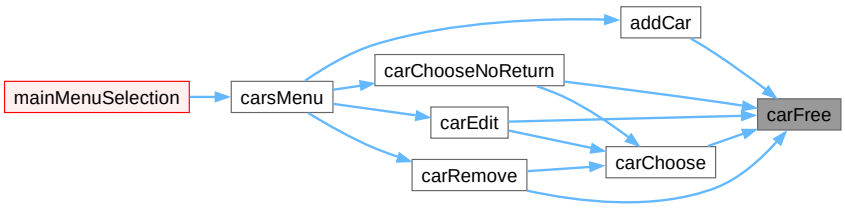
Deallocates car.

Parameters

car	Car to be freed.

Definition at line 36 of file car.c.

Here is the caller graph for this function:



5.1.2.3 carGetList()

```
struct List * carGetList (
            int sType,
            bool desc )
```

Get list of cars.

Parameters

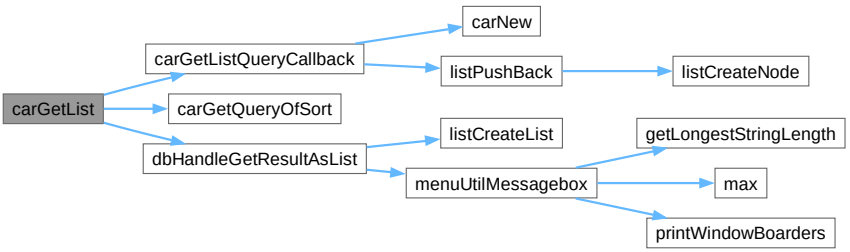
sType	Sort type corresponding to CarSort.
desc	Whether sorting should be descending. • false – ascending • true – descending

Returns

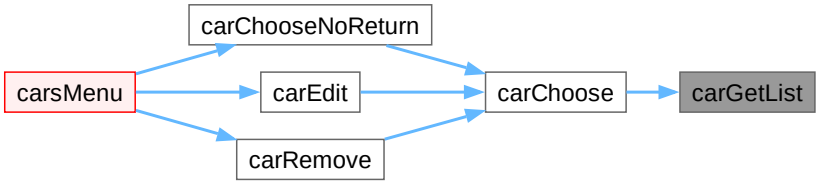
List of cars.

Definition at line 154 of file car.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.4 carGetListQueryCallback()

```
static int carGetListQueryCallback (
            struct List * list,
            int argc,
            const char ** argv,
            const char **const colNames )  [static]
```

CallBack function for dbHandleGetResultAsList. Transforms row into Car.

Parameters

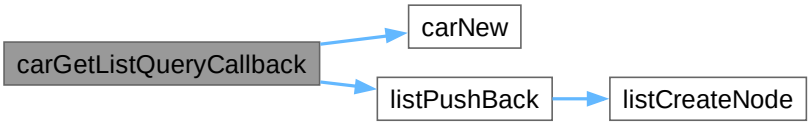
list	List to insert data into.
argc	How many columns are there.
argv	Values in columns.
colNames	names of columns.

Returns

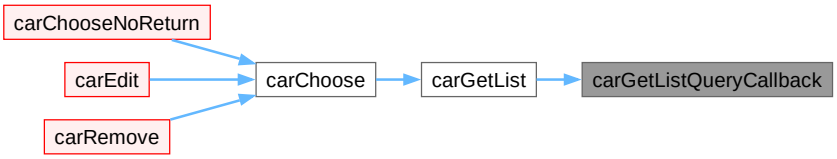
0.

Definition at line 69 of file car.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.5 carGetQueryOfSort()

```
char * carGetQueryOfSort (
            int sType,
            bool desc )
```

Generates SQL query.

Parameters

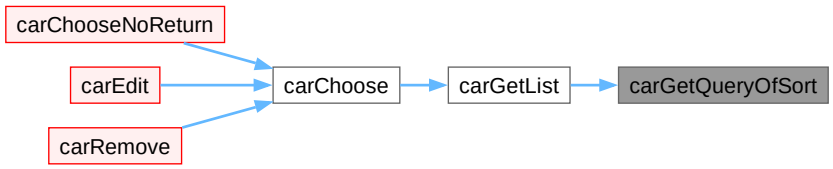
sType	CarSort based on which to sort.
desc	Whether sorting should be descending. • false – ascending • true – descending

Returns

query.

Definition at line 112 of file car.c.

Here is the caller graph for this function:



5.1.2.6 carIsComplete()

```
bool carIsComplete (
            const struct Car * car )
```

Checks if every field in Car is set.

Parameters

car	Car to check.

Returns

- True if Car is complete.
- False otherwise.

Definition at line 51 of file car.c.

Here is the caller graph for this function:

5.1.2.7 carNew()

```
struct Car * carNew ( )
```

Allocates and returns car containing nothing.

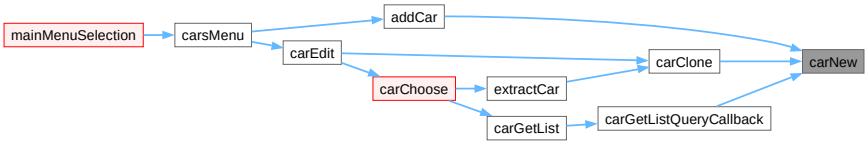
Returns

Car object.

C style strings are NULL pointers. Car::m_ID is INVALIDCARID.

Definition at line 24 of file car.c.

Here is the caller graph for this function:



5.2 car.c

Go to the documentation of this file.

```
00001 #include "car.h"
00002 #include "dbhandle.h"
00003 #include "list.h"
00004 #include "menuutil.h"
00005 #include <assert.h>
00006 #include <ncurses.h>
00007 #include <stdbool.h>
00008 #include <stdio.h>
00009 #include <stdlib.h>
00010 #include <string.h>
00011
00024 struct Car *carNew() {
00025   struct Car *result = calloc(sizeof(struct Car), 1);
00026   result->m_ID = INVALIDCARID;
00027   result->m_yOfProd = INVALIDCARYOFPROD;
00028   result->m_mileage = INVALIDCARMILEAGE;
00029   return result;
00030 }
00031
00036 void carFree(struct Car *car) {
00037   free(car->m_regNum);
00038   free(car->m_brand);
00039   free(car->m_model);
00040   free(car->m_color);
00041   free(car);
00042 }
00043
00051 bool carIsComplete(const struct Car *car) {
00052   bool result = true;
00053   if (!car || car->m_regNum == NULL || car->m_brand == NULL ||
00054       car->m_model == NULL ||car->m_yOfProd == INVALIDCARYOFPROD ||
00055       car->m_color == NULL || car->m_mileage == INVALIDCARMILEAGE)
00056     result = false;
00057   return result;
00058 }
00059
00069 static int carGetListQueryCallback(struct List *list, int argc,
00070                                    const char **argv,
00071                                    const char **const colNames) {
00072   assert(list && argv && argc && colNames);
00073   struct Car *car = carNew();
00074   for (int i = 0; i < argc; ++i) {
00075     const char *colName = colNames[i];
00076     const char *val = argv[i];
00077     if (!strcmp(colName, "ID")) {
00078       car->m_ID = atoi(val);
00079     } else if (!strcmp(colName, "regNum")) {
00080       car->m_regNum = calloc(FORMFIELDLENGTH + 1, sizeof(char));
00081       strcpy(car->m_regNum, val);
00082     } else if (!strcmp(colName, "brand")) {
00083       car->m_brand = calloc(FORMFIELDLENGTH + 1, sizeof(char));
00084       strcpy(car->m_brand, val);
```

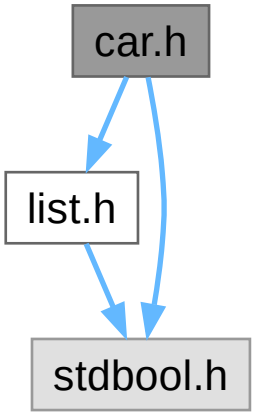
```
00085     } else if (!strcmp(colName, "model")) {
00086       car->m_model = calloc(FORMFIELDLENGTH + 1, sizeof(char));
00087       strcpy(car->m_model, val);
00088     } else if (!strcmp(colName, "yOfProd")) {
00089       car->m_yOfProd = atoi(val);
00090     } else if (!strcmp(colName, "color")) {
00091       car->m_color = calloc(FORMFIELDLENGTH + 1, sizeof(char));
00092       strcpy(car->m_color, val);
00093     } else if (!strcmp(colName, "mileage")) {
00094       car->m_mileage = atol(val);
00095     } else {
00096       fprintf(stderr, "Car to structure fail. FAILED on %s", colName);
00097       abort();
00098     }
00099   }
00100   listPushBack(list, car);
00101   return 0;
00102 }
00103
00112 char *carGetQueryOfSort(int sType, bool desc) {
00113   char *query = calloc(1000, sizeof(char));
00114   strcpy(query,
00115          "SELECT ID, regNum, brand, model, yOfProd, color, mileage FROM cars "
00116          "ORDER BY ");
00117   assert(sType >= 0 && sType < carSort_MAX);
00118   char *orderStr = NULL;
00119   switch (sType) {
00120     case carSort_regNum:
00121       orderStr = "regNum";
00122       break;
00123     case carSort_brand:
00124       orderStr = "brand";
00125       break;
00126     case carSort_model:
00127       orderStr = "model";
00128       break;
00129     case carSort_yOfProd:
00130       orderStr = "yOfProd";
00131       break;
00132     case carSort_color:
00133       orderStr = "color";
00134       break;
00135     case carSort_mileage:
00136       orderStr = "mileage";
00137       break;
00138   }
00139   strcat(query, orderStr);
00140   if (desc)
00141     strcat(query, " DESC");
00142   strcat(query, ";");
00143   return query;
00144 }
00145
00154 struct List *carGetList(int sType, bool desc) {
00155   struct List *res = NULL;
00156   char *q = carGetQueryOfSort(sType, desc);
00157   dbHandleGetResultAsList(
00158       &res, (int (*)(void *, int, char **, char **))carGetListQueryCallback,
00159       q);
00160   return res;
00161 }
00162
00168 void carClone(struct Car **dest, const struct Car *src) {
00169   struct Car *res = NULL;
00170
00171   res = carNew();
00172   res->m_ID = src->m_ID;
00173
00174   res->m_yOfProd = src->m_yOfProd;
00175
00176   res->m_mileage = src->m_mileage;
00177
00178   res->m_regNum = calloc(FORMFIELDLENGTH + 1, sizeof(char));
00179   strcpy(res->m_regNum, src->m_regNum);
00180
00181   res->m_brand = calloc(FORMFIELDLENGTH + 1, sizeof(char));
00182   strcpy(res->m_brand, src->m_brand);
00183
00184   res->m_model = calloc(FORMFIELDLENGTH + 1, sizeof(char));
00185   strcpy(res->m_model, src->m_model);
00186
00187   res->m_color = calloc(FORMFIELDLENGTH + 1, sizeof(char));
00188   strcpy(res->m_color, src->m_color);
00189
00190   *dest = res;
00191 }
00192
```

5.3 car.h File Reference

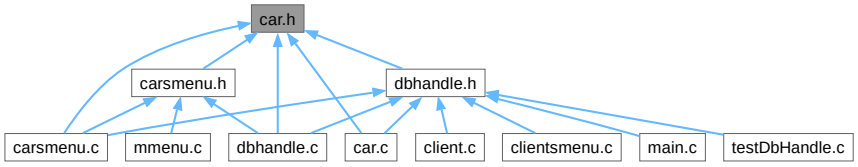
Interface for Car structure.

```
#include ¨list.h¨
#include <stdbool.h>
```

Include dependency graph for car.h:



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



Classes

- struct Car

  Structure containing information about car. More...

Macros

- #define INVALIDCARID -1

  Indicates that Car::m_ID is not valid.
- #define INVALIDCARYOFPROD -1

  Indicates that Car::m_yOfProd is not valid.
- #define INVALIDCARMILEAGE -1

  Indicates that Car::m_mileage is not valid.

Enumerations

- enum CarSort {
  carSort_regNum , carSort_brand , carSort_model , carSort_yOfProd ,
  carSort_color , carSort_mileage , carSort_MAX }

  Car sort types.

Functions

- struct Car ∗ carNew ()

  Allocates and returns car containing nothing.
- void carFree (struct Car ∗car)

  Deallocates car.
- bool carIsComplete (const struct Car ∗car)

  Checks if every field in Car is set.
- char ∗ carGetQueryOfSort (int sType, bool desc)

  Generates SQL query.
- struct List ∗ carGetList (int sType, bool desc)

  Get list of cars.
- void carClone (struct Car ∗∗dest, const struct Car ∗src)

  Make a clone of Car. It allocates memory internally.

5.3.1 Detailed Description

Interface for Car structure.

Definition in file car.h.

5.3.2 Class Documentation

5.3.2.1 struct Car

Structure containing information about car.

Definition at line 29 of file car.h.

Collaboration diagram for Car:

Class Members

char *	m_brand	Make/brand of the car.
char *	m_color	color.
int	m_ID	Car ID.
long	m_mileage	car mileage in KM.
char *	m_model	model.
char *	m_regNum	registration number.
short	m_yOfProd	Year of production.

5.3.3 Macro Definition Documentation

5.3.3.1 INVALIDCARID

```
#define INVALIDCARID -1
```

Indicates that Car::m_ID is not valid.

Definition at line 16 of file car.h.

5.3.3.2 INVALIDCARMILEAGE

```
#define INVALIDCARMILEAGE -1
```

Indicates that Car::m_mileage is not valid.

Definition at line 24 of file car.h.

5.3.3.3 INVALIDCARYOFPROD

```
#define INVALIDCARYOFPROD -1
```

Indicates that Car::m_yOfProd is not valid.

Definition at line 20 of file car.h.

5.3.4 Enumeration Type Documentation

5.3.4.1 CarSort

```
enum CarSort
```

Car sort types.

Enumerator

carSort_regNum	registration number
carSort_brand	Make/brand of the car.
carSort_model	model
carSort_yOfProd	Year of production.
carSort_color	color
carSort_mileage	car mileage in KM.
carSort_MAX	how many of CarSort types exist

Definition at line 49 of file car.h.

5.3.5 Function Documentation

5.3.5.1 carClone()

```
void carClone (
            struct Car ** dest,
            const struct Car * src )
```

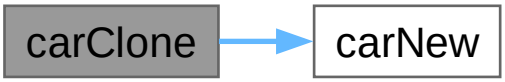
Make a clone of Car. It allocates memory internally.

Parameters

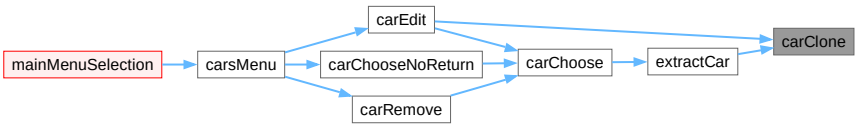
dest	Car structure where data will be cloned into.
src	Car to create a clone of.

Definition at line 168 of file car.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.5.2 carFree()

```
void carFree (
            struct Car * car )
```

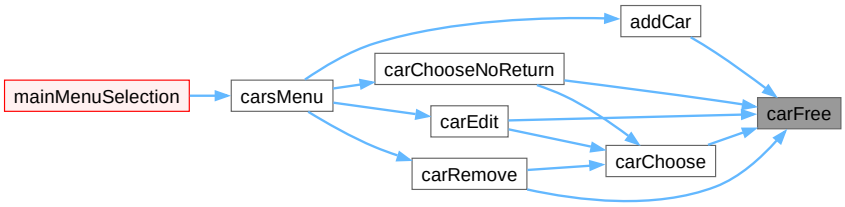
Deallocates car.

Parameters

car	Car to be freed.

Definition at line 36 of file car.c.

Here is the caller graph for this function:



5.3.5.3 carGetList()

```
struct List * carGetList (
```

```
            int sType,
            bool desc )
```

Get list of cars.

Parameters

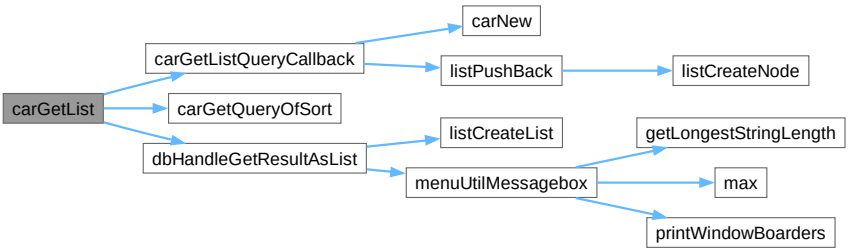
sType	Sort type corresponding to CarSort.
desc	Whether sorting should be descending. • false – ascending • true – descending

Returns

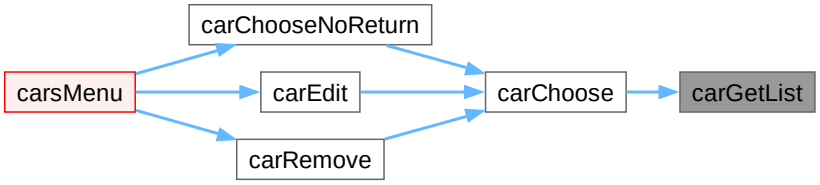
List of cars.

Definition at line 154 of file car.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.5.4 carGetQueryOfSort()

```
char * carGetQueryOfSort (
            int sType,
            bool desc )
```

Generates SQL query.

Parameters

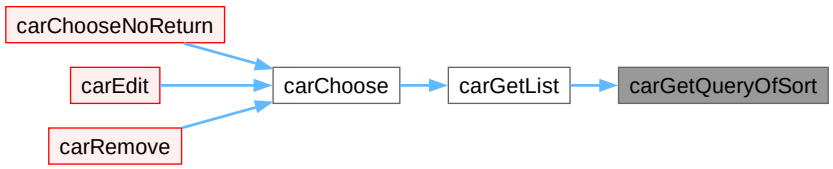
sType	CarSort based on which to sort.
desc	Whether sorting should be descending. • false – ascending • true – descending

Returns

query.

Definition at line 112 of file car.c.

Here is the caller graph for this function:



5.3.5.5 carIsComplete()

```
bool carIsComplete (
            const struct Car * car )
```

Checks if every field in Car is set.

Parameters

car	Car to check.

Returns

- True if Car is complete.
- False otherwise.

Definition at line 51 of file car.c.

Here is the caller graph for this function:

5.3.5.6 carNew()

```
struct Car * carNew ( )
```

Allocates and returns car containing nothing.

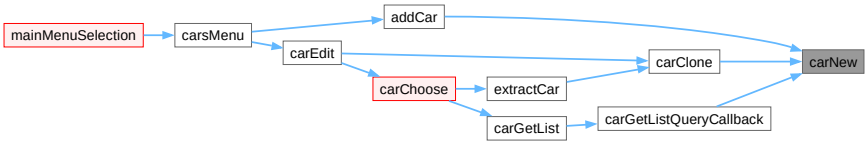
Returns

Car object.

C style strings are NULL pointers. Car::m_ID is INVALIDCARID.

Definition at line 24 of file car.c.

Here is the caller graph for this function:



5.4 car.h

Go to the documentation of this file.

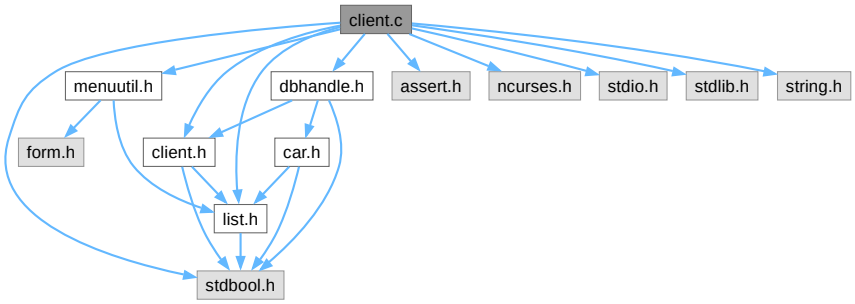
```
00001 #ifndef CAR_H_
00002 #define CAR_H_
00003
00010 #include "list.h"
00011 #include <stdbool.h>
00012
00016 #define INVALIDCARID -1
00020 #define INVALIDCARYOFPROD -1
00024 #define INVALIDCARMILEAGE -1
00025
00029 struct Car {
00031   int m_ID;
00033   char *m_regNum;
00035   char *m_brand;
00037   char *m_model;
00039   short m_yOfProd;
00041   char *m_color;
00043   long m_mileage;
00044 };
00045
00049 enum CarSort {
00051   carSort_regNum,
00053   carSort_brand,
00055   carSort_model,
00057   carSort_yOfProd,
00059   carSort_color,
00061   carSort_mileage,
00063   carSort_MAX
00064 };
00065
00066 struct Car *carNew();
00067 void carFree(struct Car *car);
00068
00069 bool carIsComplete(const struct Car *car);
00070
00071 char *carGetQueryOfSort(int sType, bool desc);
00072 struct List *carGetList(int sType, bool desc);
00073 void carClone(struct Car **dest, const struct Car *src);
00074 #endif // CAR_H_
```

5.5 client.c File Reference

Implements function to operate on client.

```
#include ¨client.h¨
#include ¨dbhandle.h¨
#include ¨list.h¨
#include ¨menuutil.h¨
#include <assert.h>
#include <ncurses.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for client.c:



Functions

- struct Client ∗ clientNew ()

  Allocates and returns client containing nothing.
- void clientFree (struct Client ∗client)

  Dealloactes client.
- bool clientIsComplete (const struct Client ∗const client)

  Checks if every field despite Client::m_ID in Client is set.
- static int clientGetListQueryCallback (struct List ∗list, int argc, const char ∗∗argv, const char ∗∗const colNames)

  CallBack function for dbHandleGetResultAsList . Transforms row into Client.
- char ∗ clientGetQueryOfSort (int sType, bool desc)

  Generates SQL query.
- struct List ∗ clientGetList (int sType, bool desc)

  Get list of clients.
- void clientClone (struct Client ∗∗dest, const struct Client ∗src)

  make Clone of Client. It allocates memory internaly.

5.5.1 Detailed Description

Implements function to operate on client.

Definition in file client.c.

5.5.2 Macro Definition Documentation

5.5.2.1 NOTRACE

```
#define NOTRACE
```

Definition at line 12 of file client.c.

5.5.3 Function Documentation

5.5.3.1 clientClone()

```
void clientClone (
            struct Client ** dest,
            const struct Client * src )
```

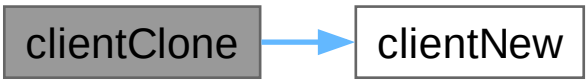
make Clone of Client. It allocates memory internaly.

Parameters

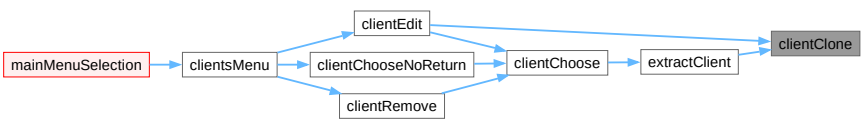
dest	Client structure where data will be cloned into.
src	Client to create clone of.

Definition at line 165 of file client.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.3.2 clientFree()

```
void clientFree (
            struct Client * client )
```

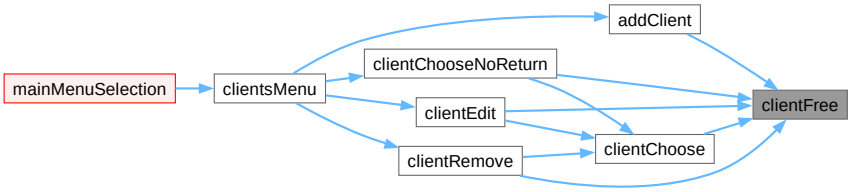
Dealloactes client.

Parameters

client	Client to be freed.

Definition at line 40 of file client.c.

Here is the caller graph for this function:



5.5.3.3 clientGetList()

```
struct List * clientGetList (
            int sType,
            bool desc )
```

Get list of clients.

Parameters

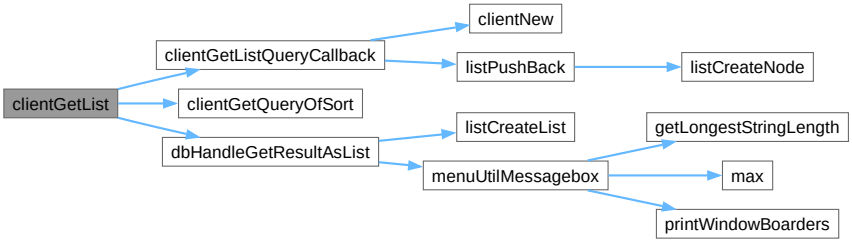
sType	Sort type coresponding to ClientSort.
desc	Wheather sorting should be descending. • flase – ascending • true – descending

Returns

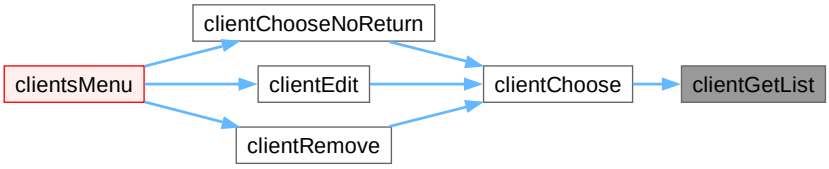
List of clients. See also Client.

Definition at line 151 of file client.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.3.4 clientGetListQueryCallback()

```
static int clientGetListQueryCallback (
             struct List * list,
             int argc,
             const char ** argv,
             const char **const colNames )  [static]
```

CallBack function for dbHandleGetResultAsList . Transforms row into Client.

Parameters

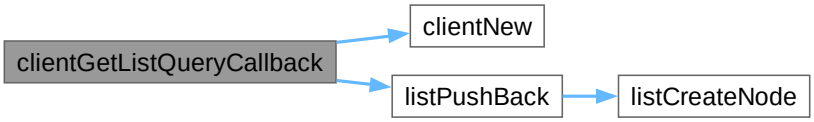
list	List to isert data into.
argc	How many columns are there.
argv	Values in columns.
colNames	names of columns.

Returns

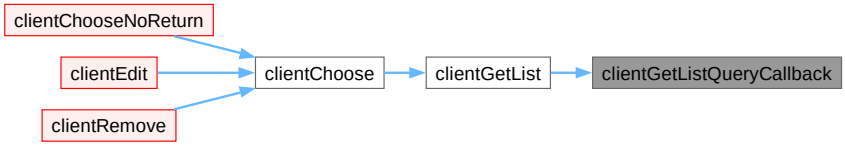
0.

Definition at line 73 of file client.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.3.5 clientGetQueryOfSort()

```
char * clientGetQueryOfSort (
            int sType,
            bool desc )
```

Generates SQL query.

Parameters

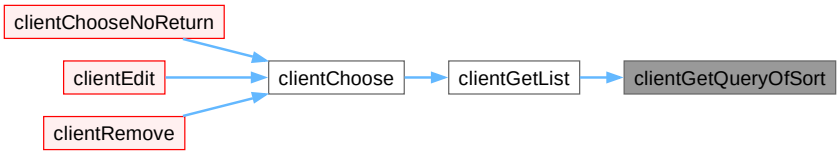
sType	ClientSort based on which to sort.
desc	Whather soring should be descending.

Returns

query.

Definition at line 112 of file client.c.

Here is the caller graph for this function:



5.5.3.6 clientIsComplete()

```
bool clientIsComplete (
            const struct Client *const client )
```

Checks if every field despite Client::m_ID in Client is set.

Parameters

client	Client to check.

Returns

- True if Client is complete.
- False otherwise.

Definition at line 54 of file client.c.

Here is the caller graph for this function:

5.5.3.7 clientNew()

```
struct Client * clientNew ( )
```

Allocates and returns client containing nothing.

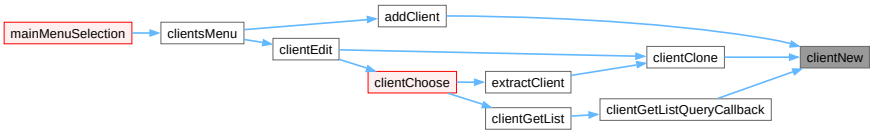
Returns

Client object.

C style strings are NULL pointers. Client::m_ID is INVALIDCLIENTID

Definition at line 26 of file client.c.

Here is the caller graph for this function:



5.6 client.c

Go to the documentation of this file.

```
00001 #include "client.h"
00002 #include "dbhandle.h"
00003 #include "list.h"
00004 #include "menuutil.h"
00005 #include <assert.h>
00006 #include <ncurses.h>
00007 #include <stdbool.h>
00008 #include <stdio.h>
00009 #include <stdlib.h>
00010 #include <string.h>
00011
00012 #define NOTRACE
00013
00026 struct Client *clientNew() {
00027   struct Client *result = calloc(sizeof(struct Client), 1);
00028   result->m_ID = INVALIDCLIENTID;
00029   result->m_cardID = INVALIDCLIENTCARDID;
00030   result->m_phoneNum = INVALIDCLIENTPHONENUM;
00031   return result;
00032 }
00033
00040 void clientFree(struct Client *client) {
00041   free(client->m_name);
00042   free(client->m_surname);
00043   free(client->m_adress);
00044   free(client);
00045 }
00046
00054 bool clientIsComplete(const struct Client *const client) {
00055   bool result = true;
00056   if (!client || client->m_adress == NULL ||
00057       client->m_cardID == INVALIDCLIENTCARDID || client->m_name == NULL ||
00058       client->m_surname == NULL || client->m_adress == NULL ||
00059       client->m_phoneNum == INVALIDCLIENTPHONENUM)
00060     result = false;
00061   return result;
00062 }
00063
00073 static int clientGetListQueryCallback(struct List *list, int argc,
00074                                       const char **argv,
00075                                       const char **const colNames) {
00076   assert(list && argv && argc && colNames);
00077   // printf("list size before push is %d\n", listSize(list));
00078   struct Client *cl = clientNew();
00079   for (int i = 0; i < argc; ++i) {
00080     const char *colName = colNames[i];
00081     const char *val = argv[i];
00082     if (!strcmp(colName, "ID")) {
00083       cl->m_ID = atoi(val);
00084     } else if (!strcmp(colName, "cardID")) {
00085       cl->m_cardID = atoi(val);
00086     } else if (!strcmp(colName, "name")) {
00087       cl->m_name = calloc(FORMFIELDLENGTH + 1, sizeof(char));
00088       strcpy(cl->m_name, val);
00089     } else if (!strcmp(colName, "surname")) {
00090       cl->m_surname = calloc(FORMFIELDLENGTH + 1, sizeof(char));
00091       strcpy(cl->m_surname, val);
00092     } else if (!strcmp(colName, "phoneNumber")) {
00093       cl->m_phoneNum = atoi(val);
00094     } else if (!strcmp(colName, "adress")) {
00095       cl->m_adress = calloc(FORMFIELDLENGTH + 1, sizeof(char));
00096       strcpy(cl->m_adress, val);
00097     } else {
00098       fprintf(stderr, "Client to structure fail. FAILED on %s", colName);
00099       abort();
00100     }
00101   }
00102   listPushBack(list, cl);
00103   return 0;
00104 }
00105
00112 char *clientGetQueryOfSort(int sType, bool desc) {
00113   char *query = calloc(1000, sizeof(char));
00114   strcpy(query,
00115          "SELECT ID, cardID, name, surname, adress, phoneNumber FROM clients "
00116          "ORDER BY ");
00117   assert(sType >= 0 && sType < clientSort_MAX);
00118   char *orderStr = NULL;
00119   switch (sType) {
00120   case clientSort_name:
00121     orderStr = "name";
00122     break;
```

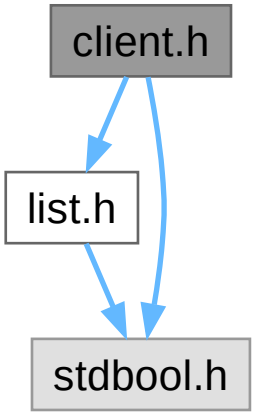
```
00123   case clientSort_cardId:
00124     orderStr = "cardID";
00125     break;
00126   case clientSort_surname:
00127     orderStr = "surname";
00128     break;
00129   case clientSort_adress:
00130     orderStr = "adress";
00131     break;
00132   case clientSort_phoneNum:
00133     orderStr = "phoneNumber";
00134     break;
00135   };
00136   strcat(query, orderStr);
00137   if (desc)
00138     strcat(query, " DESC");
00139   strcat(query, ";");
00140   return query;
00141 }
00142
00151 struct List *clientGetList(int sType, bool desc) {
00152   struct List *res = NULL;
00153   char *q = clientGetQueryOfSort(sType, desc);
00154   dbHandleGetResultAsList(
00155       &res, (int (*)(void *, int, char **, char **))clientGetListQueryCallback,
00156       q);
00157   return res;
00158 }
00159
00165 void clientClone(struct Client **dest, const struct Client *src) {
00166   struct Client *res = NULL;
00167
00168   res = clientNew();
00169   res->m_ID = src->m_ID;
00170
00171   res->m_phoneNum = src->m_phoneNum;
00172
00173   res->m_cardID = src->m_cardID;
00174
00175   res->m_adress = calloc(FORMFIELDLENGTH + 1, sizeof(char));
00176   strcpy(res->m_adress, src->m_adress);
00177
00178   res->m_name = calloc(FORMFIELDLENGTH + 1, sizeof(char));
00179   strcpy(res->m_name, src->m_name);
00180
00181   res->m_surname = calloc(FORMFIELDLENGTH + 1, sizeof(char));
00182   strcpy(res->m_surname, src->m_surname);
00183
00184   *dest = res;
00185 }
```

5.7 client.h File Reference

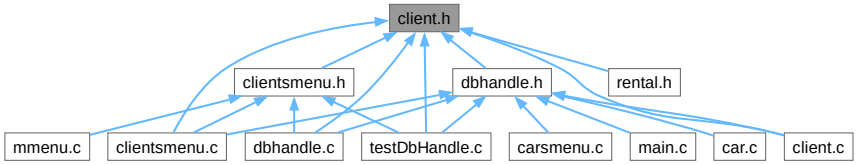
Interface for Client structure.

```
#include ¨list.h¨
#include <stdbool.h>
```

Include dependency graph for client.h:



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



Classes

- struct Client

  Structure holding client single data. More...

Macros

- #define INVALIDCLIENTID -1

  Indicates that Client::m_ID is not valid.
- #define INVALIDCLIENTCARDID -1

  Indicates that Client::m_cardID is not valid.
- #define INVALIDCLIENTPHONENUM -1

  Indicates that Client::m_phoneNum is not valid.

Enumerations

- enum ClientSort {
  clientSort_cardId , clientSort_name , clientSort_surname , clientSort_adress ,
  clientSort_phoneNum , clientSort_MAX }

  Client sort types.

Functions

- struct Client * clientNew ()
  Allocates and returns client containing nothing.
- void clientFree (struct Client *client)
  Dealloactes client.
- bool clientIsComplete (const struct Client *const client)
  Checks if every field despite Client::m_ID in Client is set.
- struct List * clientGetList (int sType, bool desc)
  Get list of clients.
- char * clientGetQueryOfSort (int sType, bool desc)
  Generates SQL query.
- void clientClone (struct Client **dest, const struct Client *src)
  make Clone of Client. It allocates memory internaly.

5.7.1 Detailed Description

Interface for Client structure.

Definition in file client.h.

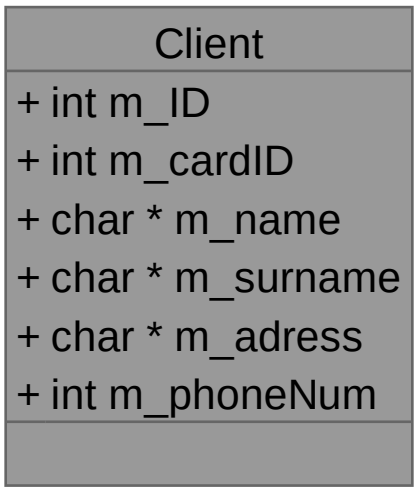
5.7.2 Class Documentation

5.7.2.1 struct Client

Structure holding client single data.

Definition at line 29 of file client.h.

Collaboration diagram for Client:



Class Members

char *	m_adress	Address.
int	m_cardID	Client's card ID.
int	m_ID	Client ID.
char *	m_name	String holding first name.
int	m_phoneNum	Phone number.
char *	m_surname	String holding second name.

5.7.3 Macro Definition Documentation

5.7.3.1 INVALIDCLIENTCARDID

```
#define INVALIDCLIENTCARDID -1
```

Indicates that Client::m_cardID is not valid.

Definition at line 20 of file client.h.

5.7.3.2 INVALIDCLIENTID

```
#define INVALIDCLIENTID -1
```

Indicates that Client::m_ID is not valid.

Definition at line 16 of file client.h.

5.7.3.3 INVALIDCLIENTPHONENUM

```
#define INVALIDCLIENTPHONENUM -1
```

Indicates that Client::m_phoneNum is not valid.

Definition at line 24 of file client.h.

5.7.4 Enumeration Type Documentation

5.7.4.1 ClientSort

```
enum ClientSort
```

Client sort types.

Enumerator

clientSort_cardId	cardId
clientSort_name	name
clientSort_surname	surname
clientSort_adress	adress
clientSort_phoneNum	phoneNum
clientSort_MAX	how many of ClientSort types exist

Definition at line 47 of file client.h.

5.7.5 Function Documentation

5.7.5.1 clientClone()

```
void clientClone (
            struct Client ** dest,
            const struct Client * src )
```

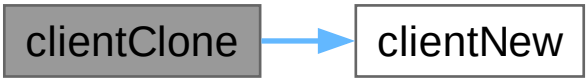
make Clone of Client. It allocates memory internaly.

Parameters

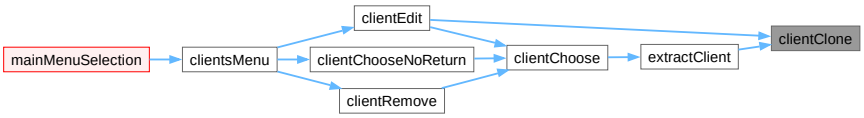
dest	Client structure where data will be cloned into.
src	Client to create clone of.

Definition at line 165 of file client.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.5.2 clientFree()

```
void clientFree (
            struct Client * client )
```

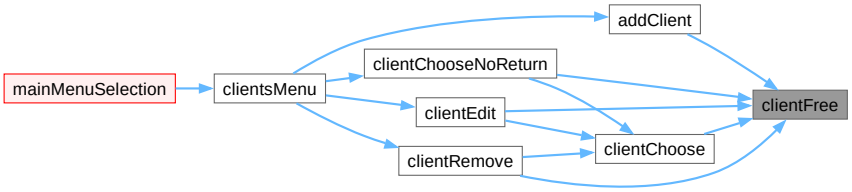
Dealloactes client.

Parameters

client	Client to be freed.

Definition at line 40 of file client.c.

Here is the caller graph for this function:



5.7.5.3 clientGetList()

```
struct List * clientGetList (
            int sType,
            bool desc )
```

Get list of clients.

Parameters

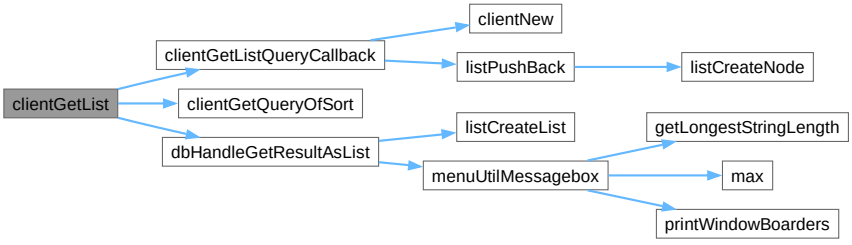
sType	Sort type coresponding to ClientSort.
desc	Wheather sorting should be descending. • flase – ascending • true – descending

Returns

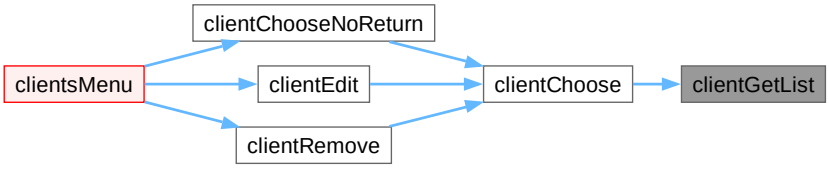
List of clients. See also Client.

Definition at line 151 of file client.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.5.4 clientGetQueryOfSort()

```
char * clientGetQueryOfSort (
            int sType,
            bool desc )
```

Generates SQL query.

Parameters

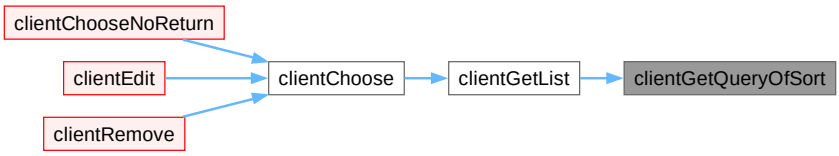
sType	ClientSort based on which to sort.
desc	Whather soring should be descending.

Returns

query.

Definition at line 112 of file client.c.

Here is the caller graph for this function:



5.7.5.5 clientIsComplete()

```
bool clientIsComplete (
            const struct Client *const client )
```

Checks if every field despite Client::m_ID in Client is set.

Parameters

client	Client to check.

Returns

- True if Client is complete.
- False otherwise.

Definition at line 54 of file client.c.

Here is the caller graph for this function:

5.7.5.6 clientNew()

```
struct Client * clientNew ( )
```

Allocates and returns client containing nothing.

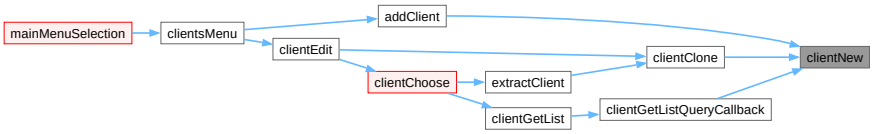
Returns

Client object.

C style strings are NULL pointers. Client::m_ID is INVALIDCLIENTID

Definition at line 26 of file client.c.

Here is the caller graph for this function:



5.8 client.h

Go to the documentation of this file.

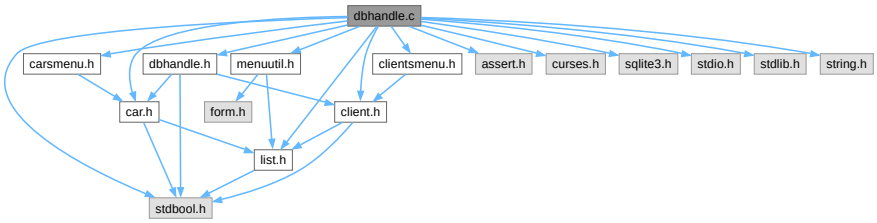
```
00001 #ifndef CLIENT_H_
00002 #define CLIENT_H_
00003
00010 #include "list.h"
00011 #include <stdbool.h>
00012
00016 #define INVALIDCLIENTID -1
00020 #define INVALIDCLIENTCARDID -1
00024 #define INVALIDCLIENTPHONENUM -1
00025
00029 struct Client {
00031   int m_ID;
00033   int m_cardID;
00035   char *m_name;
00037   char *m_surname;
00039   char *m_adress;
00041   int m_phoneNum;
00042 };
00043
00047 enum ClientSort {
00049   clientSort_cardId,
00051   clientSort_name,
00053   clientSort_surname,
00055   clientSort_adress,
00057   clientSort_phoneNum,
00059   clientSort_MAX
00060 };
00061
00062 struct Client *clientNew();
00063 void clientFree(struct Client *client);
00064
00065 bool clientIsComplete(const struct Client *const client);
00066
00067 struct List *clientGetList(int sType, bool desc);
00068 char *clientGetQueryOfSort(int sType, bool desc);
00069 void clientClone(struct Client **dest, const struct Client *src);
00070 #endif // CLIENT_H_
```

5.9 dbhandle.c File Reference

Database handling.

```
#include ¨dbhandle.h¨
#include ¨clientsmenu.h¨
#include ¨carsmenu.h¨
#include ¨list.h¨
#include <assert.h>
#include <car.h>
#include <client.h>
#include <curses.h>
#include <menuutil.h>
#include <sqlite3.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for dbhandle.c:



Functions

- bool dbHandleOpenDB ()

  Ensure that database exists.
- static bool dbHandleGetClientInsertQuery (char **out, const struct Client *client)

  Given client, get string containing SQL query that would add that client to DB.
- static bool dbHandleGetCarInsertQuery (char **out, const struct Car *car)

  Given car, get string containing SQL query that would add that car to DB.
- bool dbHandleClientInsert (const struct Client *client)

  Insert client into db.
- bool dbHandleCarInsert (const struct Car *car)

  Insert a car into the database.
- bool dbHandleGetResultAsList (struct List **out, int(*callback)(void *list, int argc, char **argv, char **colNames), const char *query)

  Given query and callback function create List based on the query.
- bool dbHandlClientRemove (int id)

  Remove clinet from database.
- bool dbHandleCarRemove (int id)

  Remove a car from the database.
- bool dbHandleClientUpdate (struct Client *toEdit)

  Update Client in database.
- bool dbHandleCarUpdate (struct Car *toEdit)

  Update a car in the database.

Variables

- static const char * DBFILENAME = ¨database.db¨

  Database path.
- static char * ENUSREDBTABLESQUERY

  SQL query ensuring tables needed for program to run exist.
- static sqlite3 * DB

  Database connection;.
- static sqlite3 * STMT

  Compiled statement for sqlite.

5.9.1 Detailed Description

Database handling.

Definition in file dbhandle.c.

5.9.2 Function Documentation

5.9.2.1 dbHandlClientRemove()

```
bool dbHandlClientRemove (
            int id )
```

Remove clinet from database.

Parameters

id	Id of clinet that will be removed

Returns

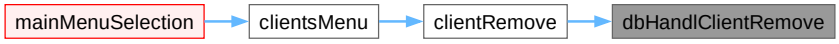
– true if success – false otherwise.

i left here

Definition at line 218 of file dbhandle.c.

Here is the call graph for this function:

Here is the caller graph for this function:



5.9.2.2 dbHandleCarInsert()

```
bool dbHandleCarInsert (
            const struct Car * car )
```

Insert a car into the database.

Parameters

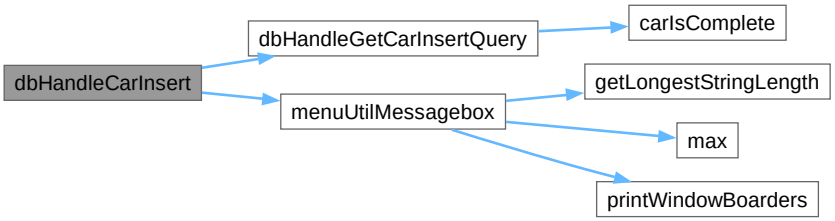
car	Car to insert into the database.

Returns

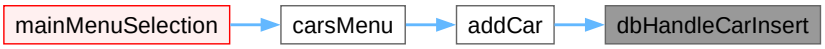
true if the car is added successfully, false if it fails.

Definition at line 161 of file dbhandle.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.2.3 dbHandleCarRemove()

```
bool dbHandleCarRemove (
            int id )
```

Remove a car from the database.

Parameters

id	ID of the car to be removed.

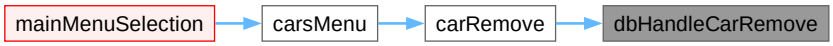
Returns

true if successful, false otherwise.

Definition at line 244 of file dbhandle.c.

Here is the call graph for this function:

Here is the caller graph for this function:



5.9.2.4 dbHandleCarUpdate()

```
bool dbHandleCarUpdate (
            struct Car * toEdit )
```

Update a car in the database.

Parameters

toEdit	Car to update.

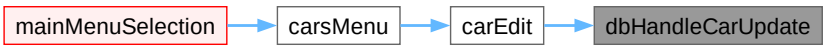
Returns

true if successful, false otherwise.

Definition at line 298 of file dbhandle.c.

Here is the call graph for this function:

Here is the caller graph for this function:



5.9.2.5 dbHandleClientInsert()

```
bool dbHandleClientInsert (
            const struct Client * client )
```

Insert client into db.

Parameters

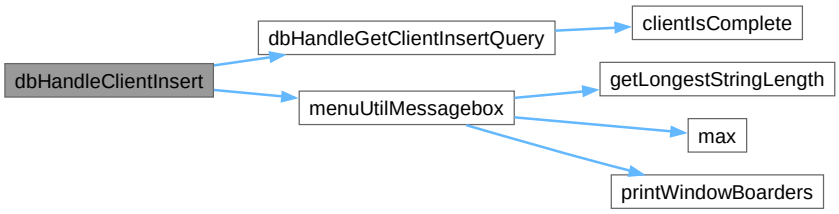
client	Client to insert into db.

Returns

- true if added client sucessfuly.
- false if failed.

Definition at line 137 of file dbhandle.c.

Here is the call graph for this function:



Here is the caller graph for this function:

5.9.2.6 dbHandleClientUpdate()

```
bool dbHandleClientUpdate (
            struct Client * toEdit )
```

Update Client in database.

Parameters

toEdit	Client to update.

Returns

Whather succeded

- true – sucess
- false – failed

Definition at line 269 of file dbhandle.c.

Here is the call graph for this function:

Here is the caller graph for this function:

5.9.2.7 dbHandleGetCarInsertQuery()

```
static bool dbHandleGetCarInsertQuery (
            char ** out,
            const struct Car * car )  [static]
```

Given car, get string containing SQL query that would add that car to DB.

Parameters

out	Where to save the query.
car	Car based on which the statement is generated.

Returns

true if the statement is created successfully, false otherwise.

Definition at line 117 of file dbhandle.c.

Here is the call graph for this function:

Here is the caller graph for this function:

5.9.2.8 dbHandleGetClientInsertQuery()

```
static bool dbHandleGetClientInsertQuery (
            char ** out,
            const struct Client * client )  [static]
```

Given client, get string containing SQL query that would add that client to DB.

Parameters

out	Where to save query.

Warning

This function does not allocate space for out.

Parameters

client	Client based on which statemnet is generated.

Returns

- true if created statement succesfully
- false otherwise.

Definition at line 95 of file dbhandle.c.

Here is the call graph for this function:

Here is the caller graph for this function:

5.9.2.9 dbHandleGetResultAsList()

```
bool dbHandleGetResultAsList (
            struct List ** out,
            int(*)(void *list, int argc, char **argv, char **colNames) callback,
            const char * query )
```

Given query and callback function create List based on the query.

Parameters

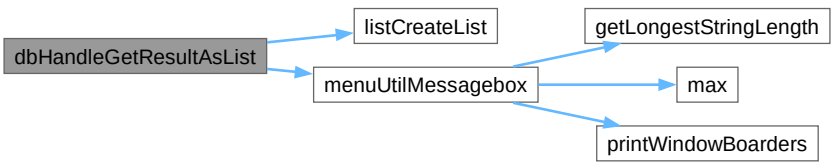
out	Where result List is stored.
callback	Function insering ListNode into List based on data returned from sqlite. First parameter is pointer to List.
query	Query to run in order to recive data.

Returns

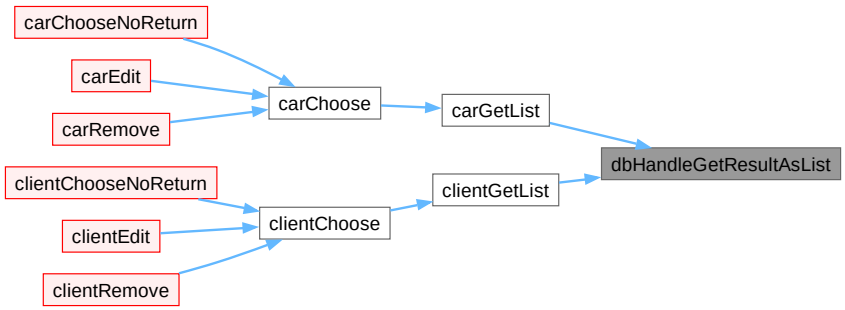
- true if sucess.
- false otherwise.

Definition at line 189 of file dbhandle.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.2.10 dbHandleOpenDB()

```
bool dbHandleOpenDB ( )
```

Ensure that database exists.

Opens connection with database.

Returns

False if ok. True if failed. Ensures that database exists and has required tables.

Definition at line 71 of file dbhandle.c.

Here is the caller graph for this function:

5.9.3 Variable Documentation

5.9.3.1 DB

```
sqlite3* DB  [static]
```

Database connection;.

Definition at line 59 of file dbhandle.c.

5.9.3.2 DBFILENAME

```
const char* DBFILENAME = ¨database.db¨  [static]
```

Database path.

Definition at line 24 of file dbhandle.c.

5.9.3.3 ENUSREDBTABLESQUERY

```
char* ENUSREDBTABLESQUERY  [static]
```

Initial value:

```
= "CREATE TABLE IF NOT EXISTS cars("
                                    "    ID  INTEGER NOT NULL,"
                                    "    regNum  TEXT,"
                                    "    brand   TEXT,"
                                    "    model   TEXT,"
                                    "    yOfProd INTEGER,"
                                    "    color   TEXT,"
                                    "    mileage INTEGER,"
                                    "   PRIMARY KEY(ID AUTOINCREMENT));"
                                    "CREATE TABLE IF NOT EXISTS 'clients' ("
                                    "   'ID' INTEGER NOT NULL UNIQUE,"
                                    "   'cardID' INTEGER,"
                                    "   'name'   TEXT,"
                                    "   'surname'    TEXT,"
                                    "   'adress'   TEXT,"
                                    "   'phoneNumber'    INTEGER,"
                                    "   PRIMARY KEY('ID' AUTOINCREMENT)"
                                    "   );"
                                    "CREATE TABLE IF NOT EXISTS 'rentals' ("
                                    "   'ID' INTEGER NOT NULL,"
                                    "   'clientID'   TEXT NOT NULL,"
                                    "   'carID'  TEXT NOT NULL,"
                                    "   'since'  TEXT,"
                                    "   'until'  TEXT,"
                                    "   PRIMARY KEY('ID' AUTOINCREMENT)"
                                    "   );"
```

SQL query ensuring tables needed for program to run exist.

Definition at line 29 of file dbhandle.c.

5.9.3.4 STMT

```
sqlite3* STMT  [static]
```

Compiled statement for sqlite.

Definition at line 64 of file dbhandle.c.

5.10 dbhandle.c

Go to the documentation of this file.

```
00001 #include "dbhandle.h"
00002 #include "clientsmenu.h"
00003 #include "carsmenu.h"
00004 #include "list.h"
00005 #include <assert.h>
00006 #include <car.h>
00007 #include <client.h>
00008 #include <curses.h>
00009 #include <menuutil.h>
00010 #include <sqlite3.h>
00011 #include <stdbool.h>
00012 #include <stdio.h>
00013 #include <stdlib.h>
00014 #include <string.h>
00015
00024 static const char *DBFILENAME = "database.db";
00025
00029 static char *ENUSREDBTABLESQUERY = "CREATE TABLE IF NOT EXISTS cars("
00030                                    "    ID  INTEGER NOT NULL,"
00031                                    "    regNum  TEXT,"
00032                                    "    brand   TEXT,"
00033                                    "    model   TEXT,"
00034                                    "    yOfProd INTEGER,"
00035                                    "    color   TEXT,"
00036                                    "    mileage INTEGER,"
00037                                    "   PRIMARY KEY(ID AUTOINCREMENT));"
00038                                    "CREATE TABLE IF NOT EXISTS 'clients' ("
00039                                    "   'ID' INTEGER NOT NULL UNIQUE,"
00040                                    "   'cardID' INTEGER,"
00041                                    "   'name'   TEXT,"
00042                                    "   'surname'    TEXT,"
00043                                    "   'adress'   TEXT,"
00044                                    "   'phoneNumber'    INTEGER,"
00045                                    "   PRIMARY KEY('ID' AUTOINCREMENT)"
00046                                    "   );"
00047                                    "CREATE TABLE IF NOT EXISTS 'rentals' ("
00048                                    "   'ID' INTEGER NOT NULL,"
00049                                    "   'clientID'   TEXT NOT NULL,"
00050                                    "   'carID'  TEXT NOT NULL,"
00051                                    "   'since'  TEXT,"
00052                                    "   'until'  TEXT,"
00053                                    "   PRIMARY KEY('ID' AUTOINCREMENT)"
00054                                    "   );";
00055
00059 static sqlite3 *DB;
00060
00064 static sqlite3 *STMT;
00065
00071 bool dbHandleOpenDB() {
00072   sqlite3_open(DBFILENAME, &DB); // open database
00073   char *err;
00074   int rc = sqlite3_exec(DB, ENUSREDBTABLESQUERY, NULL, NULL, &err);
00075   // if failed return true
00076   if (rc != SQLITE_OK) {
00077     fprintf(stderr, "%s\n", err);
00078     sqlite3_free(err);
00079     return true;
00080   }
00081   sqlite3_close(DB);
00082   return false;
00083 }
00084
00095 static bool dbHandleGetClientInsertQuery(char **out,
00096                                          const struct Client *client) {
00097   assert(*out);
00098   if (!clientIsComplete(client)) {
00099     return false;
00100   }
00101
00102   sprintf(*out,
00103           "INSERT INTO clients (cardID, name, surname, adress, phoneNumber) "
00104           "VALUES (%d, '%s', '%s', '%s', %d);",
00105           client->m_cardID, client->m_name, client->m_surname, client->m_adress,
00106           client->m_phoneNum);
00107   return true;
00108 }
00109
00117 static bool dbHandleGetCarInsertQuery(char **out, const struct Car *car) {
00118   assert(*out);
00119   if (!carIsComplete(car)) {
00120     return false;
00121   }
```

```
00122
00123   sprintf(*out,
00124           "INSERT INTO cars (regNum, brand, model, yOfProd, color, mileage) "
00125           "VALUES ('%s', '%s', '%s', %d, '%s', %ld);",
00126           car->m_regNum, car->m_brand, car->m_model, car->m_yOfProd, car->m_color, car->m_mileage);
00127   return true;
00128 }
00129
00137 bool dbHandleClientInsert(const struct Client *client) {
00138   sqlite3_open(DBFILENAME, &DB); // open
00139   char *err = NULL;
00140   char *query = calloc(500, sizeof(char));
00141   bool status = true;
00142   if (dbHandleGetClientInsertQuery(&query, client)) {
00143     int rc = sqlite3_exec(DB, query, NULL, NULL, &err);
00144     if (rc != SQLITE_OK) {
00145       const char *msg[] = {err, NULL};
00146       menuUtilMessagebox("dbHandleClientInsert failed", msg);
00147       sqlite3_free(err);
00148     }
00149   }
00150   free(query);
00151   sqlite3_close(DB);
00152   return true;
00153 }
00154
00161 bool dbHandleCarInsert(const struct Car *car) {
00162   sqlite3_open(DBFILENAME, &DB);
00163   char *err = NULL;
00164   char *query = calloc(500, sizeof(char));
00165   bool status = true;
00166   if (dbHandleGetCarInsertQuery(&query, car)) {
00167     int rc = sqlite3_exec(DB, query, NULL, NULL, &err);
00168     if (rc != SQLITE_OK) {
00169       const char *msg[] = {err, NULL};
00170       menuUtilMessagebox("dbHandleCarInsert failed", msg);
00171       sqlite3_free(err);
00172     }
00173   }
00174   free(query);
00175   sqlite3_close(DB);
00176   return true;
00177 }
00178
00189 bool dbHandleGetResultAsList(struct List **out,
00190                              int (*callback)(void *list, int argc, char **argv,
00191                                              char **colNames),
00192                              const char *query) {
00193   if (SQLITE_OK != sqlite3_open(DBFILENAME, &DB)) {
00194     abort();
00195   }
00196   char *err = NULL;
00197   bool status = true;
00198   // if passed list is null ptr acllocate it.
00199   if (*out == NULL)
00200     *out = listCreateList();
00201   int rc = sqlite3_exec(DB, query, callback, *out, &err);
00202   if (rc != SQLITE_OK) {
00203     const char *msg[] = {err, NULL};
00204     menuUtilMessagebox("dbHandleGetResultAsList", msg);
00205     sqlite3_free(err);
00206     status = 0;
00207   }
00208   sqlite3_close(DB);
00209   return status;
00210 }
00211
00218 bool dbHandlClientRemove(int id) {
00219   sqlite3_open(DBFILENAME, &DB); // open
00220   char *err = NULL;
00221   char *query = calloc(500, sizeof(char));
00223   sprintf(query, "DELETE FROM clients WHERE clients.ID=%d", id);
00224
00225   bool status = true;
00226   int rc = sqlite3_exec(DB, query, NULL, NULL, &err);
00227   if (rc != SQLITE_OK) {
00228     const char *msg[] = {err, NULL};
00229     menuUtilMessagebox("dbHandleRemoveClient", msg);
00230     sqlite3_free(err);
00231     status = false;
00232   }
00233   // free(query); // ??
00234   sqlite3_close(DB);
00235   return status;
00236 }
00237
00244 bool dbHandleCarRemove(int id) {
```

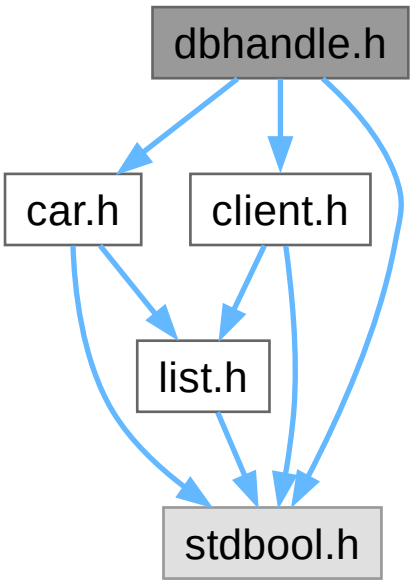
```
00245   sqlite3_open(DBFILENAME, &DB);
00246   char *err = NULL;
00247   char *query = calloc(500, sizeof(char));
00248   sprintf(query, "DELETE FROM cars WHERE cars.ID=%d", id);
00249
00250   bool status = true;
00251   int rc = sqlite3_exec(DB, query, NULL, NULL, &err);
00252   if (rc != SQLITE_OK) {
00253     const char *msg[] = {err, NULL};
00254     menuUtilMessagebox("dbHandleCarRemove", msg);
00255     sqlite3_free(err);
00256     status = false;
00257   }
00258   sqlite3_close(DB);
00259   return status;
00260 }
00261
00269 bool dbHandleClientUpdate(struct Client *toEdit) {
00270   char *err = NULL;
00271   char *query = calloc(700, sizeof(char));
00272   sprintf(query,
00273           "UPDATE clients SET cardID = '%d', name = '%s', surname "
00274           "='%s', adress='%s', phoneNumber='%d' WHERE ID = %d;",
00275           toEdit->m_cardID, toEdit->m_name, toEdit->m_surname, toEdit->m_adress,
00276           toEdit->m_phoneNum, toEdit->m_ID);
00277   bool status = true;
00278   sqlite3_open(DBFILENAME, &DB); // open
00279   int rc = sqlite3_exec(DB, query, NULL, NULL, &err);
00280   if (rc != SQLITE_OK) {
00281     const char *msg[] = {err, NULL};
00282     assert(err);
00283     menuUtilMessagebox("dbHandleClientUpdate ERROR", msg);
00284     sqlite3_free(err);
00285     status = false;
00286   }
00287   // free(query); // ??
00288   sqlite3_close(DB);
00289   return status;
00290 }
00291
00298 bool dbHandleCarUpdate(struct Car *toEdit) {
00299   char *err = NULL;
00300   char *query = calloc(700, sizeof(char));
00301   sprintf(query,
00302           "UPDATE cars SET regNum = '%s', brand = '%s', model = '%s', "
00303           "yOfProd = %d, color = '%s', mileage = %ld WHERE ID = %d;",
00304           toEdit->m_regNum, toEdit->m_brand, toEdit->m_model, toEdit->m_yOfProd,
00305           toEdit->m_color, toEdit->m_mileage, toEdit->m_ID);
00306   bool status = true;
00307   sqlite3_open(DBFILENAME, &DB);
00308   int rc = sqlite3_exec(DB, query, NULL, NULL, &err);
00309   if (rc != SQLITE_OK) {
00310     const char *msg[] = {err, NULL};
00311     assert(err);
00312     menuUtilMessagebox("dbHandleCarUpdate ERROR", msg);
00313     sqlite3_free(err);
00314     status = false;
00315   }
00316   sqlite3_close(DB);
00317   return status;
00318 }
```

5.11 dbhandle.h File Reference

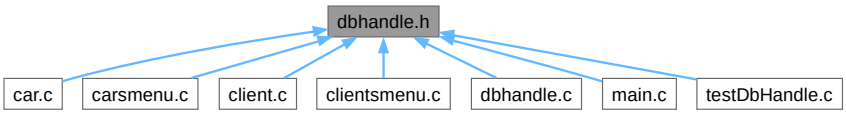
Database operations interface.

```
#include "car.h"
#include "client.h"
#include <stdbool.h>
```

Include dependency graph for dbhandle.h:



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



Functions

- bool dbHandleOpenDB ()

  Opens connection with database.
- static bool **dbHandleGetClientInsertQuery** (char **out, const struct Client *client)
- static bool **dbHandleGetCarInsertQuery** (char **out, const struct Car *car)
- bool dbHandleClientInsert (const struct Client *client)

  Insert client into db.
- bool dbHandleCarInsert (const struct Car *car)

  Insert a car into the database.
- bool dbHandleGetResultAsList (struct List **out, int(*callback)(void *list, int argc, char **argv, char **colNames), const char *query)

  Given query and callback function create List based on the query.
- bool dbHandlClientRemove (int id)

  Remove clinet from database.
- bool dbHandleCarRemove (int id)

Remove a car from the database.

- bool dbHandleClientUpdate (struct Client ∗toEdit)

  Update Client in database.

- bool dbHandleCarUpdate (struct Car ∗toEdit)

  Update a car in the database.

5.11.1 Detailed Description

Database operations interface.

Definition in file dbhandle.h.

5.11.2 Function Documentation

5.11.2.1 dbHandlClientRemove()

```
bool dbHandlClientRemove (
            int id )
```

Remove clinet from database.

Parameters

id	Id of clinet that will be removed

Returns

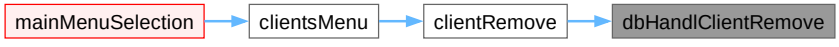
– true if success – false otherwise.

i left here

Definition at line 218 of file dbhandle.c.

Here is the call graph for this function:

Here is the caller graph for this function:



5.11.2.2 dbHandleCarInsert()

```
bool dbHandleCarInsert (
            const struct Car * car )
```

Insert a car into the database.

Parameters

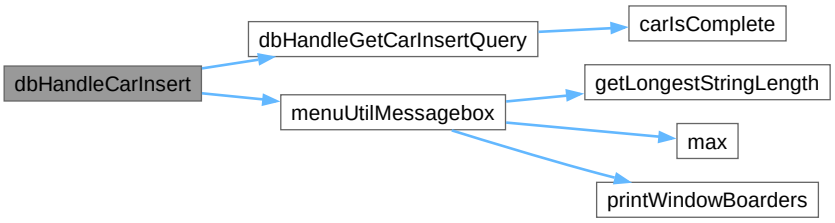
car	Car to insert into the database.

Returns

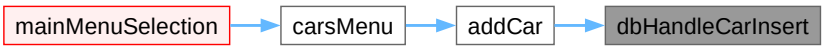
true if the car is added successfully, false if it fails.

Definition at line 161 of file dbhandle.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.2.3 dbHandleCarRemove()

```
bool dbHandleCarRemove (
            int id )
```

Remove a car from the database.

Parameters

id	ID of the car to be removed.

Returns

true if successful, false otherwise.

Definition at line 244 of file dbhandle.c.

Here is the call graph for this function:

Here is the caller graph for this function:

5.11.2.4 dbHandleCarUpdate()

```
bool dbHandleCarUpdate (
            struct Car * toEdit )
```

Update a car in the database.

Parameters

toEdit	Car to update.

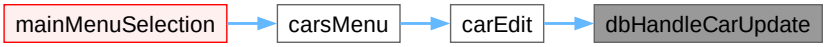
Returns

true if successful, false otherwise.

Definition at line 298 of file dbhandle.c.

Here is the call graph for this function:

Here is the caller graph for this function:



5.11.2.5 dbHandleClientInsert()

```
bool dbHandleClientInsert (
            const struct Client * client )
```

Insert client into db.

Parameters

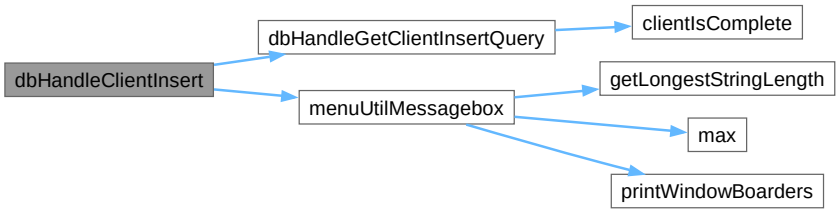
client	Client to insert into db.

Returns

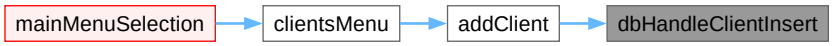
- true if added client sucessfuly.
- false if failed.

Definition at line 137 of file dbhandle.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.2.6 dbHandleClientUpdate()

```
bool dbHandleClientUpdate (
            struct Client * toEdit )
```

Update Client in database.

Parameters

toEdit	Client to update.

Returns

Whather succeded

- true – sucess
- false – failed

Definition at line 269 of file dbhandle.c.

Here is the call graph for this function:

Here is the caller graph for this function:

5.11.2.7 dbHandleGetResultAsList()

```
bool dbHandleGetResultAsList (
            struct List ** out,
            int(*)(void *list, int argc, char **argv, char **colNames) callback,
            const char * query )
```

Given query and callback function create List based on the query.

Parameters

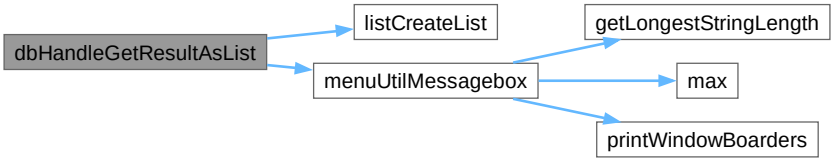
out	Where result List is stored.
callback	Function insering ListNode into List based on data returned from sqlite. First parameter is pointer to List.
query	Query to run in order to recive data.

Returns

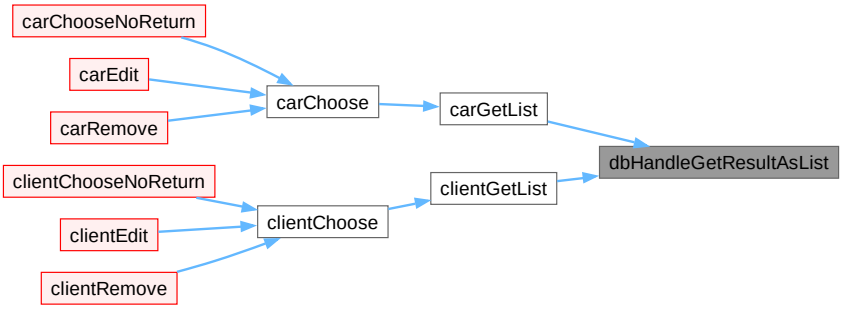
- true if sucess.
- false otherwise.

Definition at line 189 of file dbhandle.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.2.8 dbHandleOpenDB()

```
bool dbHandleOpenDB ( )
```

Opens connection with database.

Returns

False if ok. True if failed. Ensures that database exists and has required tables.

Opens connection with database.

Returns

False if ok. True if failed. Ensures that database exists and has required tables.

Definition at line 71 of file dbhandle.c.

Here is the caller graph for this function:

5.12 dbhandle.h

Go to the documentation of this file.

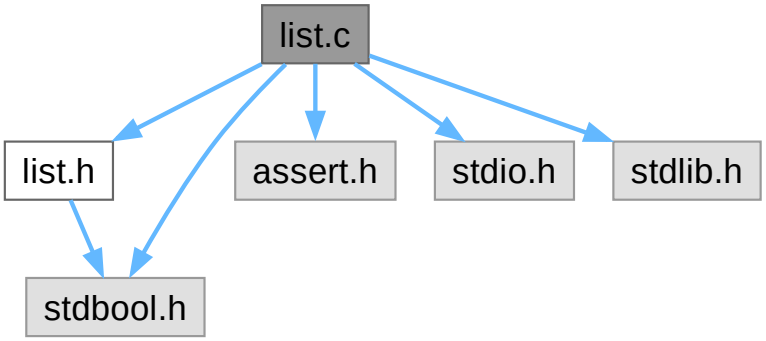
```
00001 #ifndef DBHANDLE_H
00002 #define DBHANDLE_H
00003 #include "car.h"
00004 #include "client.h"
00005 #include <stdbool.h>
00016 bool dbHandleOpenDB();
00017
00018 static bool dbHandleGetClientInsertQuery(char **out, const struct Client *client);
00019 static bool dbHandleGetCarInsertQuery(char **out, const struct Car *car);
00020
00021 bool dbHandleClientInsert(const struct Client *client);
00022 bool dbHandleCarInsert(const struct Car *car);
00023
00024 bool dbHandleGetResultAsList(struct List **out,
00025                              int (*callback)(void *list, int argc, char **argv,
00026                                              char **colNames),
00027                              const char *query);
00028
00029 bool dbHandlClientRemove(int id);
00030 bool dbHandleCarRemove(int id);
00031
00032 bool dbHandleClientUpdate(struct Client *toEdit);
00033 bool dbHandleCarUpdate(struct Car *toEdit);
00034
00035 #endif // DBHANDLE_H
```

5.13 list.c File Reference

Doubly linked list implementation.

```
#include ¨list.h¨
#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
```

Include dependency graph for list.c:



Functions

- struct ListNode * listCreateNode (void *data)

  Instantiates ListNode containing pointer to data.

- bool listInsertBefore (struct List *list, struct ListNode *node, void *data)
  Inserts data into list before given node.
- bool listDealocateListNode (struct ListNode *node)
  Free memory taken by ListNode.
- struct List * listCreateList ()
  Returns empty list.
- bool listPushFront (struct List *list, void *data)
  Adds item at the front of list.
- bool listPushBack (struct List *list, void *data)
  Adds item at the end of list.
- bool listInsert (struct List *list, void *data, bool(*prevFun)(const void *, const void *))
  Inserts data in list at appropriate positon so that list remains sorted.
- struct ListNode * listGetFront (struct List *list)
  Returns pointer to the first element of list.
- struct ListNode * listGetBack (struct List *list)
  Returns pointer to the last element of list.
- bool listDeleteNode (struct List *list, struct ListNode *node)
  Removes ListNode from List.
- int listSize (const struct List *const list)
  How many elements are there in List.

5.13.1 Detailed Description

Doubly linked list implementation.

Definition in file list.c.

5.13.2 Function Documentation

5.13.2.1 listCreateList()

```
struct List * listCreateList ( )
```

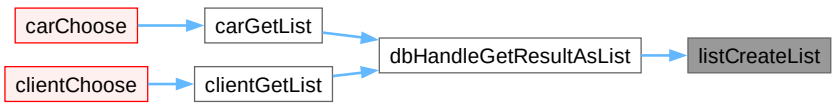
Returns empty list.

Returns

Poitner to empty list.

Definition at line 45 of file list.c.

Here is the caller graph for this function:



5.13.2.2 listCreateNode()

```
struct ListNode * listCreateNode (
            void * data )
```

Instantiates ListNode containing pointer to data.

Parameters

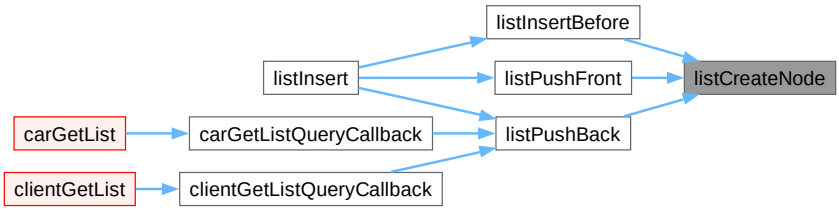
data	Pointer to data.

Returns

Pointer to ListNode that was instatntitated.

Definition at line 124 of file list.c.

Here is the caller graph for this function:



5.13.2.3 listDealocateListNode()

```
bool listDealocateListNode (
            struct ListNode * node )
```

Free memory taken by ListNode.

Parameters

node	For removal.

Returns

False if succeed.

Definition at line 34 of file list.c.

Here is the caller graph for this function:

5.13.2.4 listDeleteNode()

```
bool listDeleteNode (
            struct List * list,
            struct ListNode * node )
```

Removes ListNode from List.

Parameters

list	List from which ListNode has to be deleted.
node	ListNode for removal.

Returns

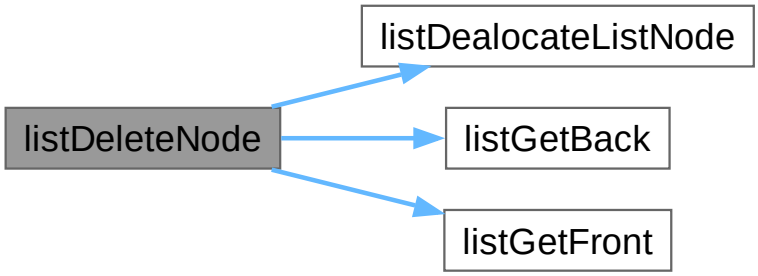
False if deleted successfully.

Warning

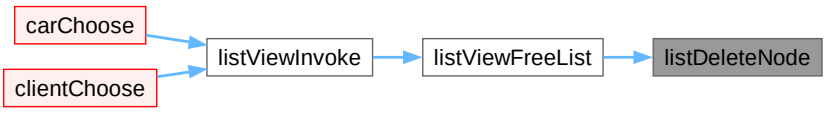
Does not remove data allocated by user in ListNode::m_data.

Definition at line 175 of file list.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.13.2.5 listGetBack()

```
struct ListNode * listGetBack (
            struct List * list )
```

Returns pointer to the last element of list.

Parameters

list	List pointer of which last element is wanted.

Returns

Pointer to the last element in the List.

- Returns NULL if List is empty.

Definition at line 165 of file list.c.

Here is the caller graph for this function:

5.13.2.6 listGetFront()

```
struct ListNode * listGetFront (
            struct List * list )
```

Returns pointer to the first element of list.

Parameters

list	List pointer of which first element is wanted.

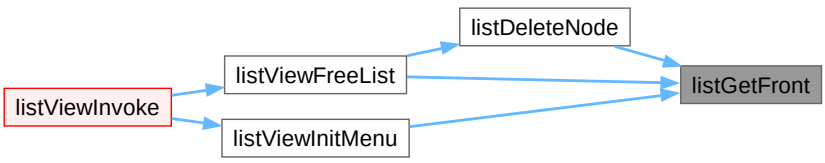
Returns

Pointer to the first element in the List.

- Returns NULL if List is empty.

Definition at line 157 of file list.c.

Here is the caller graph for this function:



5.13.2.7 listInsert()

```
bool listInsert (
            struct List * list,
            void * data,
            bool(*)(const void *, const void *) prevFun )
```

Inserts data in list at appropriate positon so that list remains sorted.

Parameters

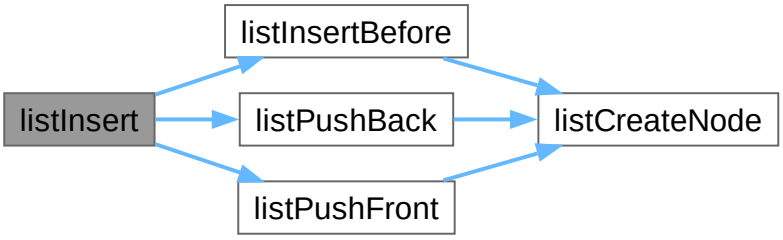
list	List into which item is inserted.
data	Pointer to data that will be inserted.
prevFun	Pointer to function that compares two data instances.

Returns

false if everything is fine.

Definition at line 106 of file list.c.

Here is the call graph for this function:



5.13.2.8 listInsertBefore()

```
bool listInsertBefore (
            struct List * list,
            struct ListNode * node,
            void * data )
```

Inserts data into list before given node.

Parameters

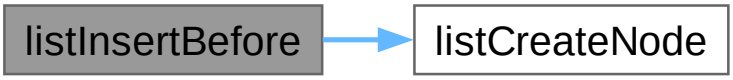
list	List pointer.
node	Node before which data should be inserted.
data	Data to be inserted pointer.

Returns

False if everything is fine. List has to be non empty;

Definition at line 133 of file list.c.

Here is the call graph for this function:



Here is the caller graph for this function:

5.13.2.9 listPushBack()

```
bool listPushBack (
            struct List * list,
            void * data )
```

Adds item at the end of list.

Parameters

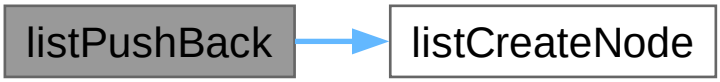
list	List into which item is added.
data	Pointer to data that will be pushed.

Returns

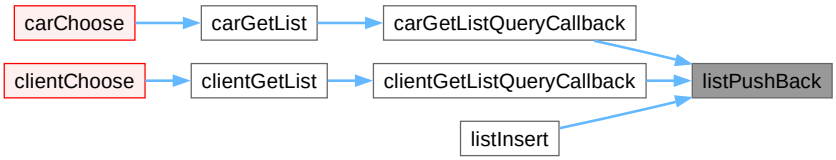
false if everything is fine.

Definition at line 82 of file list.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.13.2.10 listPushFront()

```
bool listPushFront (
            struct List * list,
            void * data )
```

Adds item at the front of list.

Parameters

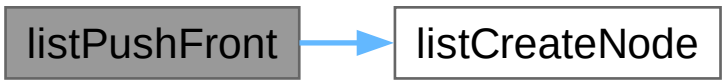
list	List into which item is added.
data	Pointer to data that will be pushed.

Returns

false if everything is fine.

Definition at line 61 of file list.c.

Here is the call graph for this function:



Here is the caller graph for this function:

5.13.2.11 listSize()

```
int listSize (
            const struct List *const list )
```

How many elements are there in List.

Parameters

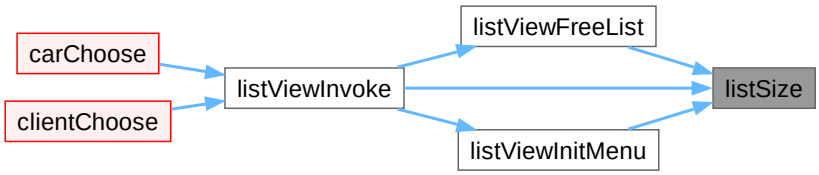
list	List of which size is to be retrived.

Returns

Size of list.

Definition at line 205 of file list.c.

Here is the caller graph for this function:



5.14 list.c

Go to the documentation of this file.

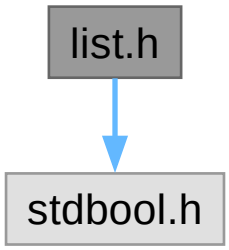
```
00001 #include "list.h"
00002 #include <assert.h>
00003 #include <stdbool.h>
00004 #include <stdio.h>
00005 #include <stdlib.h>
00006
00017 struct ListNode *listCreateNode(void *data);
00018
00027 bool listInsertBefore(struct List *list, struct ListNode *node, void *data);
00028
00034 bool listDealocateListNode(struct ListNode *node) {
00035   assert(node != NULL);
00036   // free(node->m_data);
00037   free(node);
00038   return false;
00039 }
00040
00045 struct List *listCreateList() {
00046   struct List *list = malloc(sizeof(struct List));
00047   // ensure that memory was allocated
00048   assert(list);
00049   list->m_front = NULL;
00050   list->m_back = NULL;
00051   list->m_size = 0;
00052   return list;
00053 };
00054
00061 bool listPushFront(struct List *list, void *data) {
00062   struct ListNode *node = listCreateNode(data);
00063   // if List is empty
00064   if (list->m_back == NULL) {
00065     list->m_back = node;
00066   } else {
00067     // Make old first node previous of new first node.
00068     node->m_next = list->m_front;
00069     node->m_next->m_prev = node;
00070   }
00071   list->m_front = node;
00072   ++list->m_size;
00073   return false;
00074 }
00075
00082 bool listPushBack(struct List *list, void *data) {
00083
00084   struct ListNode *node = listCreateNode(data);
00085   // if List is empty
00086   if (list->m_front == NULL) {
00087     list->m_front = node;
00088   } else {
00089     // Make old last node previous of new last node.
00090     node->m_prev = list->m_back;
00091     node->m_prev->m_next = node;
00092   }
00093   list->m_back = node;
00094   ++list->m_size;
00095   return false;
00096 }
00097
00106 bool listInsert(struct List *list, void *data,
00107                 bool (*prevFun)(const void *, const void *)) {
```

```
00108   struct ListNode *it = list->m_front;
00109   // go to next element as long as cureent element has to be after new element
00110   while (it != NULL && prevFun(it->m_data, data)) {
00111     it = it->m_next;
00112   }
00113   // if first is what we look for or list is empty
00114   if (it == list->m_front)
00115     listPushFront(list, data);
00116   else if (it == NULL)
00117     listPushBack(list, data);
00118   else {
00119     listInsertBefore(list, it, data);
00120   }
00121   return true;
00122 }
00123
00124 struct ListNode *listCreateNode(void *data) {
00125   struct ListNode *node = malloc(sizeof(struct ListNode));
00126   assert(node);
00127   node->m_data = data;
00128   node->m_next = NULL;
00129   node->m_prev = NULL;
00130   return node;
00131 }
00132
00133 bool listInsertBefore(struct List *list, struct ListNode *node, void *data) {
00134   // Ensure list is non empty.
00135   assert(list->m_front != NULL);
00136   struct ListNode *newNode = listCreateNode(data);
00137   // Make newNode previous point to node that is previous to the node that we
00138   // insert berfore.
00139   newNode->m_prev = node->m_prev;
00140   // Make node that we insert before next of newNode.
00141   newNode->m_next = node;
00142
00143   // Make node preceeding newNode point to newNode.
00144   newNode->m_prev->m_next = newNode;
00145   // Make node after newNode point to newNode.
00146   node->m_prev = newNode;
00147   ++list->m_size;
00148   return false;
00149 }
00150
00157 struct ListNode *listGetFront(struct List *list) { return list->m_front; }
00158
00165 struct ListNode *listGetBack(struct List *list) { return list->m_back; }
00166
00175 bool listDeleteNode(struct List *list, struct ListNode *node) {
00176   assert(node != NULL);
00177   // If node for removal is first in the list
00178   if (node == listGetFront(list)) {
00179     list->m_front = node->m_next;
00180     // If it's only one ListNode in List.
00181     if (node == listGetBack(list))
00182       list->m_back = node->m_prev;
00183     else
00184       node->m_next->m_prev = node->m_prev;
00185   }
00186   // ListNode for removal is not at front in the List.
00187   else {
00188     node->m_prev->m_next = node->m_next;
00189     // node for removal is last in the list
00190     if (node == listGetBack(list))
00191       list->m_back = node->m_prev;
00192     else
00193       node->m_next->m_prev = node->m_prev;
00194   }
00195   --list->m_size;
00196   listDealocateListNode(node);
00197   return false;
00198 }
00199
00205 int listSize(const struct List *const list) { return list->m_size; }
```

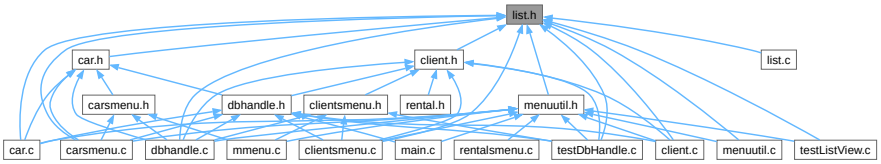
5.15 list.h File Reference

Doubly linked list interface.

```
#include ¨stdbool.h¨
```
Include dependency graph for list.h:



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



Classes

- struct ListNode

  Doubly linked list node. More...
- struct List

  Stores pointers to first and last elements of double linked list. More...

Functions

- struct List ∗ listCreateList ()

  Returns empty list.
- bool listPushFront (struct List ∗list, void ∗data)

  Adds item at the front of list.
- bool listPushBack (struct List ∗list, void ∗data)

  Adds item at the end of list.
- bool listInsert (struct List ∗list, void ∗data, bool(∗prevFun)(const void ∗, const void ∗))

  Inserts data in list at appropriate positon so that list remains sorted.
- struct ListNode ∗ listGetFront (struct List ∗list)

  Returns pointer to the first element of list.
- struct ListNode ∗ listGetBack (struct List ∗list)

  Returns pointer to the last element of list.
- bool listDeleteNode (struct List ∗list, struct ListNode ∗node)

  Removes ListNode from List.
- int listSize (const struct List ∗const list)

  How many elements are there in List.

5.15.1 Detailed Description

Doubly linked list interface.

Todo move function documentation into .c file.

Definition in file list.h.

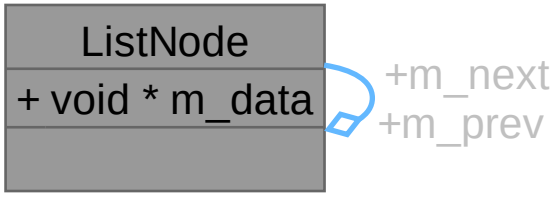
5.15.2 Class Documentation

5.15.2.1 struct ListNode

Doubly linked list node.

Definition at line 16 of file list.h.

Collaboration diagram for ListNode:



Class Members

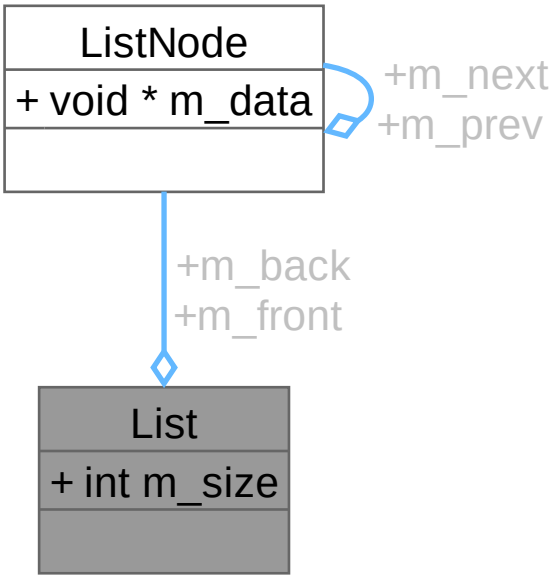
void *	m_data	Pointer to data.
struct ListNode *	m_next	Pointer to the next element of the list. • NULL if it's last element.
struct ListNode *	m_prev	Pointer to the previous element of the list. • NULL if it's first element.

5.15.2.2 struct List

Stores pointers to first and last elements of double linked list.

Definition at line 30 of file list.h.

Collaboration diagram for List:



Class Members

struct ListNode *	m_back	Pointer to last elemenet of the list.
struct ListNode *	m_front	Pointer to first elemenet of the list.
int	m_size	Number of elements in the list.

5.15.3 Function Documentation

5.15.3.1 listCreateList()

```
struct List * listCreateList ( )
```

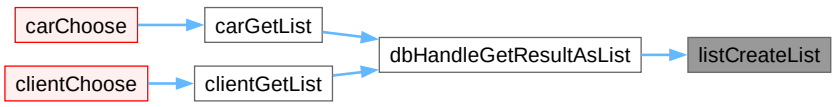
Returns empty list.

Returns

Poitner to empty list.

Definition at line 45 of file list.c.

Here is the caller graph for this function:



5.15.3.2 listDeleteNode()

```
bool listDeleteNode (
            struct List * list,
            struct ListNode * node )
```

Removes ListNode from List.

Parameters

list	List from which ListNode has to be deleted.
node	ListNode for removal.

Returns

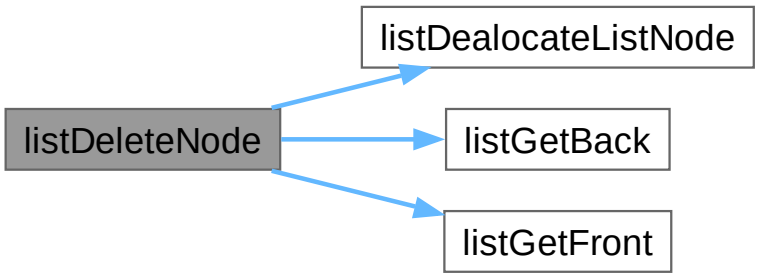
False if deleted successfully.

Warning

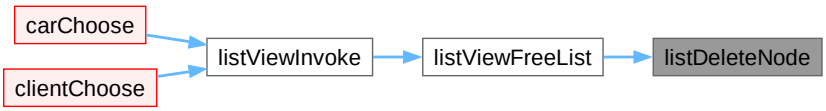
Does not remove data allocated by user in ListNode::m_data.

Definition at line 175 of file list.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.15.3.3 listGetBack()

```
struct ListNode * listGetBack (
            struct List * list )
```

Returns pointer to the last element of list.

Parameters

list	List pointer of which last element is wanted.

Returns

Pointer to the last element in the List.

- Returns NULL if List is empty.

Definition at line 165 of file list.c.

Here is the caller graph for this function:

5.15.3.4 listGetFront()

```
struct ListNode * listGetFront (
            struct List * list )
```

Returns pointer to the first element of list.

Parameters

list	List pointer of which first element is wanted.

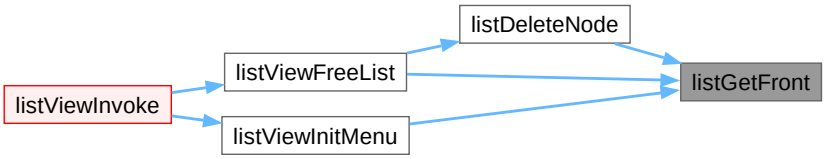
Returns

Pointer to the first element in the List.

- Returns NULL if List is empty.

Definition at line 157 of file list.c.

Here is the caller graph for this function:



5.15.3.5 listInsert()

```
bool listInsert (
            struct List * list,
            void * data,
            bool(*)(const void *, const void *) prevFun )
```

Inserts data in list at appropriate positon so that list remains sorted.

Parameters

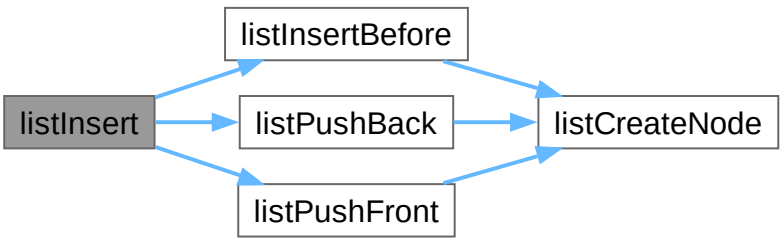
list	List into which item is inserted.
data	Pointer to data that will be inserted.
prevFun	Pointer to function that compares two data instances.

Returns

false if everything is fine.

Definition at line 106 of file list.c.

Here is the call graph for this function:



5.15.3.6 listPushBack()

```
bool listPushBack (
            struct List * list,
            void * data )
```

Adds item at the end of list.

Parameters

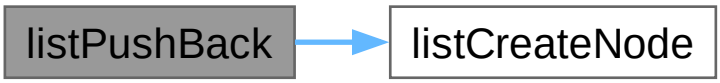
list	List into which item is added.
data	Pointer to data that will be pushed.

Returns

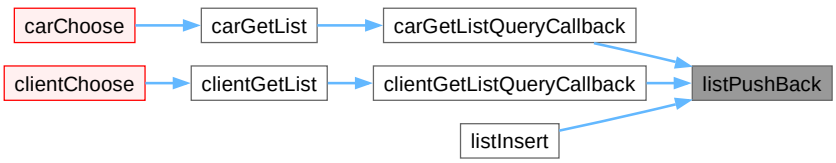
false if everything is fine.

Definition at line 82 of file list.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.15.3.7 listPushFront()

```
bool listPushFront (
            struct List * list,
            void * data )
```

Adds item at the front of list.

Parameters

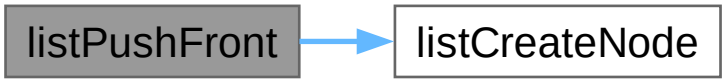
list	List into which item is added.
data	Pointer to data that will be pushed.

Returns

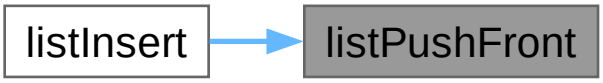
false if everything is fine.

Definition at line 61 of file list.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.15.3.8 listSize()

```
int listSize (
            const struct List *const list )
```

How many elements are there in List.

Parameters

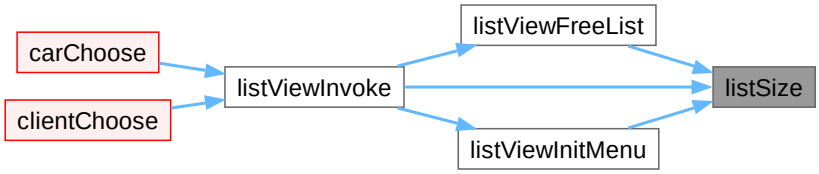
list	List of which size is to be retrived.

Returns

Size of list.

Definition at line 205 of file list.c.

Here is the caller graph for this function:



5.16 list.h

Go to the documentation of this file.

```
00001 #ifndef LIST_H
00002 #define LIST_H
00003
00004 #include "stdbool.h"
00005
00007
00016 struct ListNode {
00018   void *m_data;
00021   struct ListNode *m_prev;
00024   struct ListNode *m_next;
00025 };
00026
00030 struct List {
00032   struct ListNode *m_front;
00033
00035   struct ListNode *m_back;
00036
00038   int m_size;
00039 };
00040
00041 struct List *listCreateList();
00042
00043 bool listPushFront(struct List *list, void *data);
00044
00045 bool listPushBack(struct List *list, void *data);
00046
00047 bool listInsert(struct List *list, void *data,
00048                 bool (*prevFun)(const void *, const void *));
00049
00050 struct ListNode *listGetFront(struct List *list);
00051
00052 struct ListNode *listGetBack(struct List *list);
00053
00054 bool listDeleteNode(struct List *list, struct ListNode *node);
00055
00056 int listSize(const struct List *const list);
00057
00058 #endif // LIST_H
```

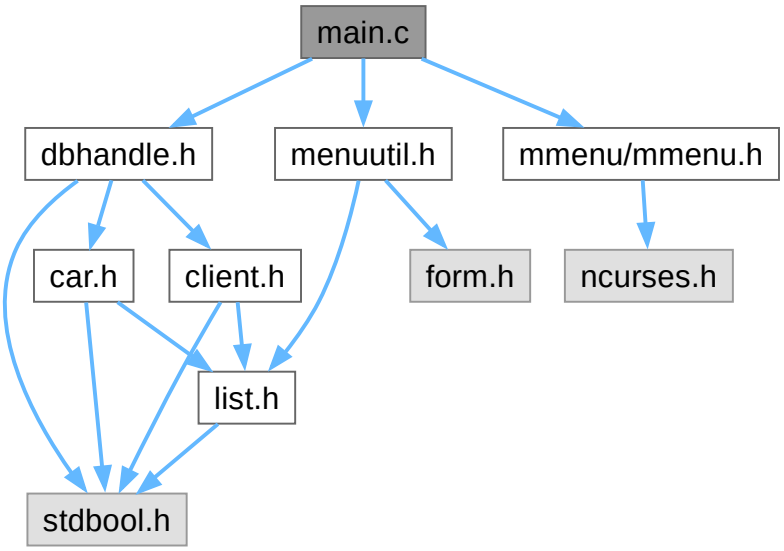
5.17 main.c File Reference

Main file.

```
#include ¨dbhandle.h¨
#include ¨menuutil.h¨
```

```
#include ¨mmenu/mmenu.h¨
```

Include dependency graph for main.c:



Functions

- int main ()

  main.

5.17.1 Detailed Description

Main file.

Definition in file main.c.

5.17.2 Function Documentation

5.17.2.1 main()

```
int main ( )
```

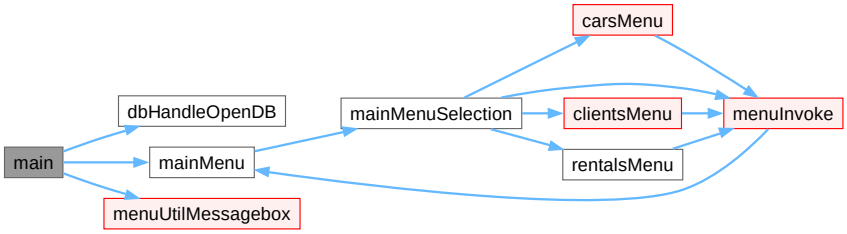
main.

Returns

0;

Definition at line 13 of file main.c.

Here is the call graph for this function:



5.18 main.c

Go to the documentation of this file.

```
00001 #include "dbhandle.h"
00002 #include "menuutil.h"
00003 #include "mmenu/mmenu.h"
00004
00013 int main() {
00014   const char *test[] = {"test", NULL};
00015   menuUtilMessagebox("TEST in main", test);
00016   if (!dbHandleOpenDB())
00017     mainMenu();
00018 }
```

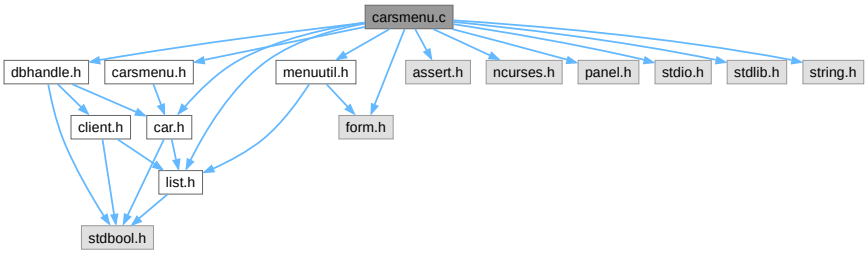
5.19 carsmenu.c File Reference

Cars menu implementation.

```
#include ¨carsmenu.h¨
#include ¨car.h¨
#include ¨dbhandle.h¨
#include ¨list.h¨
#include ¨menuutil.h¨
#include <assert.h>
#include <form.h>
#include <ncurses.h>
#include <panel.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
```

Include dependency graph for carsmenu.c:



Functions

- static bool carFormParse (struct Car **result, FORM *form)

  Parse form.
- static bool carFormEdit (struct Car **result, const struct Car *const placeHolder)

  Function for editing car.
- void addCar (void)

  Function for adding a car.
- char * carGetListViewString (const struct Car *car)

  Given car, generates a string representing car data in listView friendly format (whole row).
- static void extractCar (struct Car **out, const struct ListNode *node)

  Extracts car from the given ListNode.
- static struct Car * carChoose (void)

  Invokes the listView of cars.
- void carRemove (void)

  Removes a car.
- void carChooseNoReturn (void)

  Wrapper around carChoose frees extracted car instanntly.
- void carEdit (void)

  Edit car.
- void carsMenu (void)

  Handles displaying of cars menu.

5.19.1 Detailed Description

Cars menu implementation.

Definition in file carsmenu.c.

5.19.2 Macro Definition Documentation

5.19.2.1 NOTRACE

```
#define NOTRACE
```

Definition at line 14 of file carsmenu.c.

5.19.3 Function Documentation

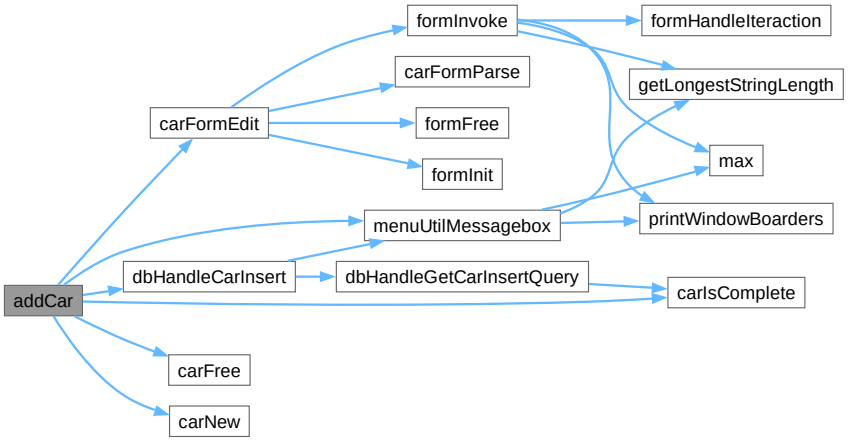
5.19.3.1 addCar()

```
void addCar (
            void  )
```

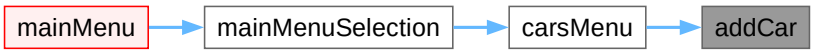
Function for adding a car.

Definition at line 132 of file carsmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.19.3.2 carChoose()

```
static struct Car * carChoose (
            void  )  [static]
```

Invokes the listView of cars.

Returns

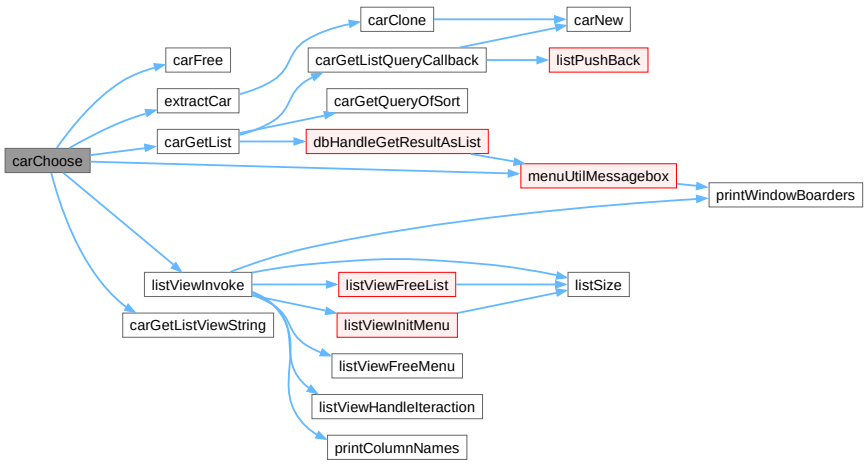
- Chosen car clone
- NULL if user cancelled.

Note

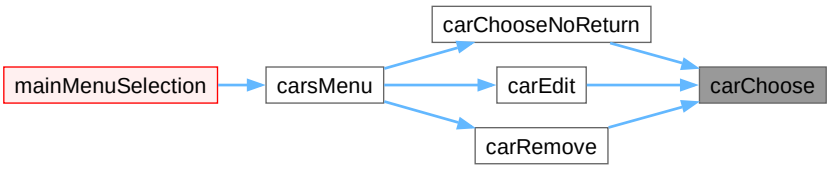
Extracted client has to be freed manually. See also clientChooseNoReturn.

Definition at line 185 of file carsmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:



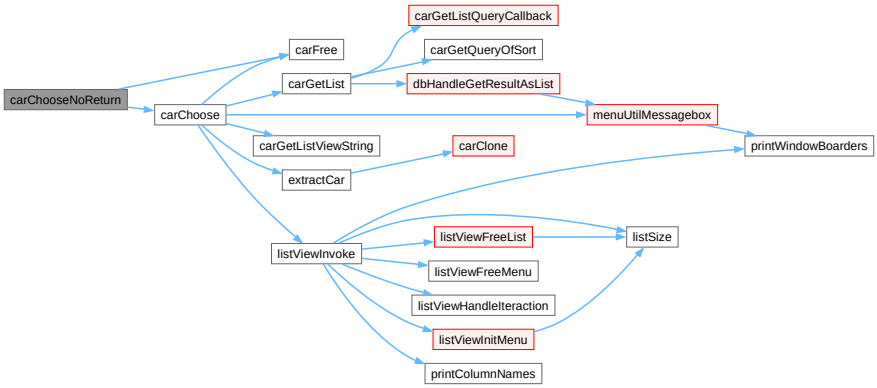
5.19.3.3 carChooseNoReturn()

```
void carChooseNoReturn (
            void  )
```

Wrapper around carChoose frees extracted car instanntly.

Definition at line 232 of file carsmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:

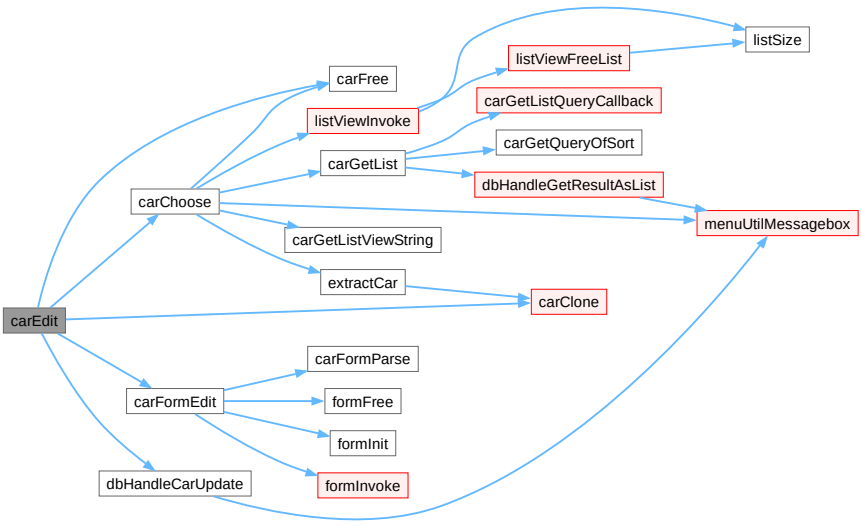
5.19.3.4 carEdit()

```
void carEdit (
              void  )
```

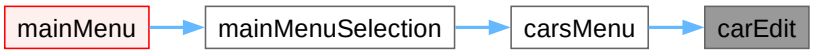
Edit car.

Definition at line 241 of file carsmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.19.3.5 carFormEdit()

```
static bool carFormEdit (
            struct Car ** result,
            const struct Car *const placeHolder )  [static]
```

Function for editing car.

Parameters

result	Where to put results at.
placeHolder	Structure of placeholder values for form. If NULL placeholders are empty (useful when instead of editing adding).

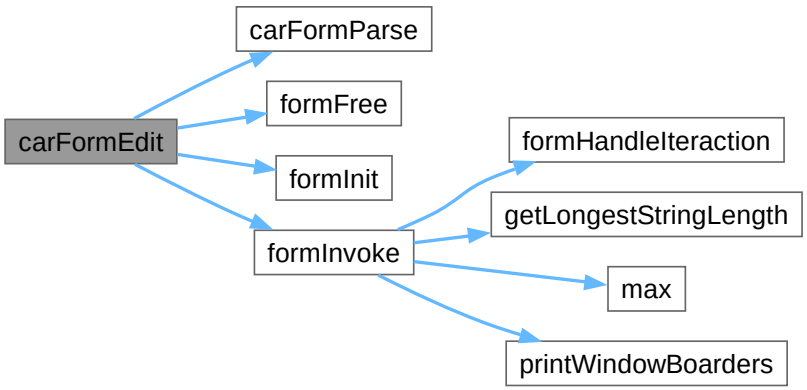
Returns

Whenever changes should be propagated.

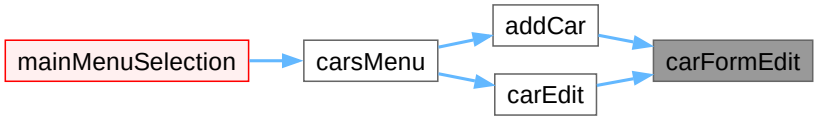
- false = nothing to do.

Definition at line 83 of file carsmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.19.3.6 carFormParse()

```
static bool carFormParse (
            struct Car ** result,
            FORM * form )  [static]
```

Parse form.

Parameters

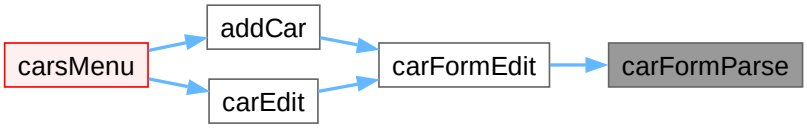
result	Car object where to save parsed result, Car::m_ID will be set to INVALIDCARID. Does not allocate object.
form	Filled already form containing car data.

Returns

- True if any of fields has been altered
- False if none of fields has been altered.

Definition at line 30 of file carsmenu.c.

Here is the caller graph for this function:



5.19.3.7 carGetListViewString()

```
char * carGetListViewString (
            const struct Car * car )
```

Given car, generates a string representing car data in listView friendly format (whole row).

Parameters

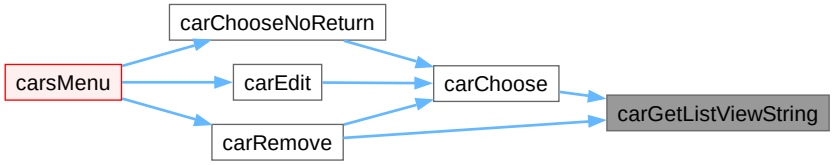
car	Car based on which to generate the string.

Returns

String representing the car.

Definition at line 152 of file carsmenu.c.

Here is the caller graph for this function:



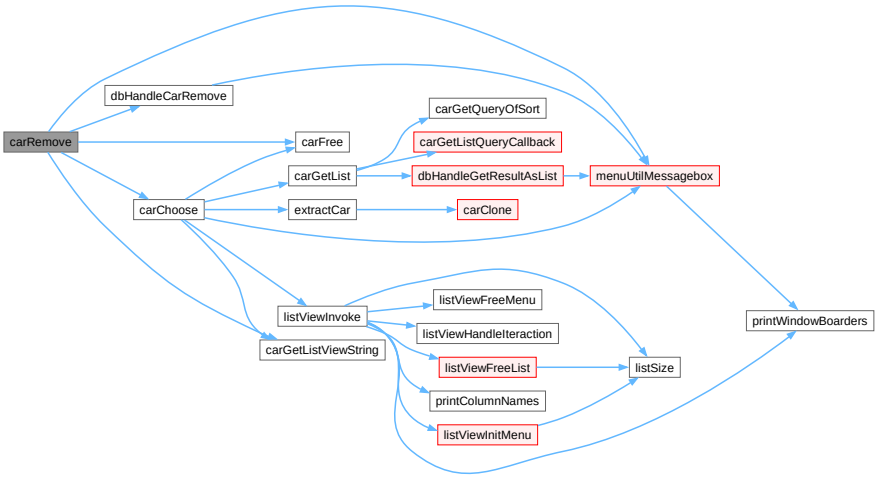
5.19.3.8 carRemove()

```
void carRemove (
            void  )
```

Removes a car.

Definition at line 214 of file carsmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:

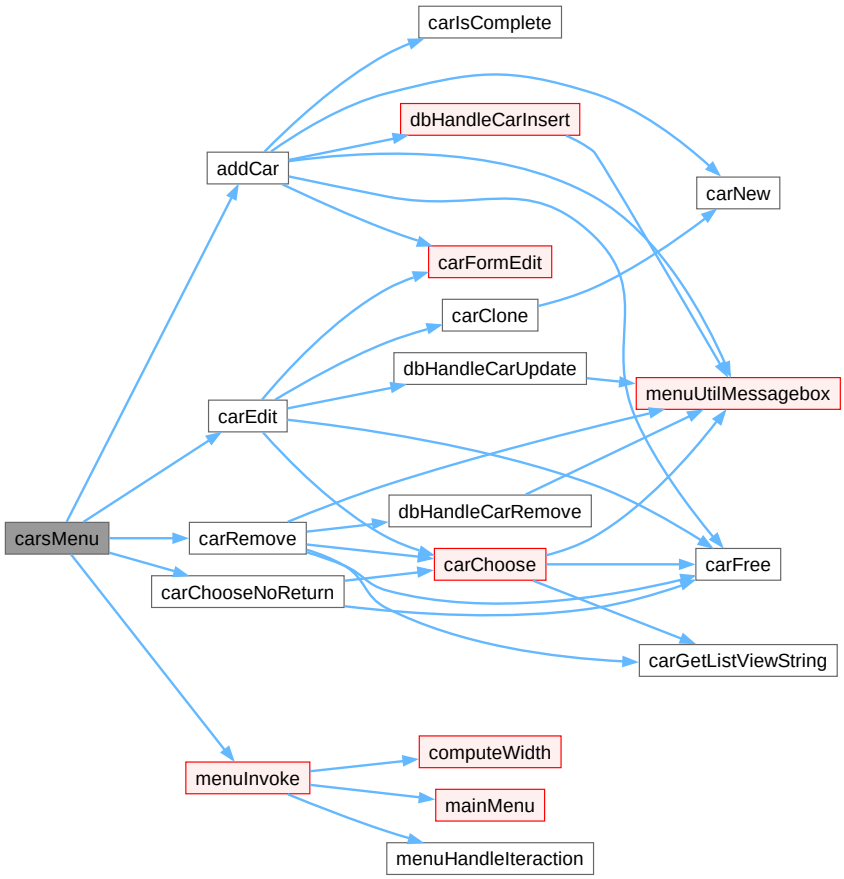
5.19.3.9 carsMenu()

```
void carsMenu (
            void  )
```

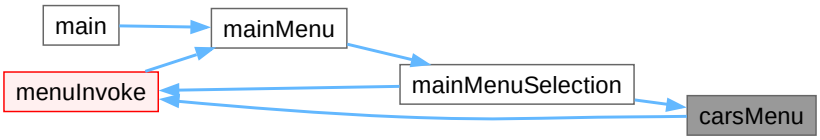
Handles displaying of cars menu.

Definition at line 257 of file carsmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.19.3.10 extractCar()

```
static void extractCar (
            struct Car ** out,
            const struct ListNode * node )  [static]
```

Extracts car from the given ListNode.

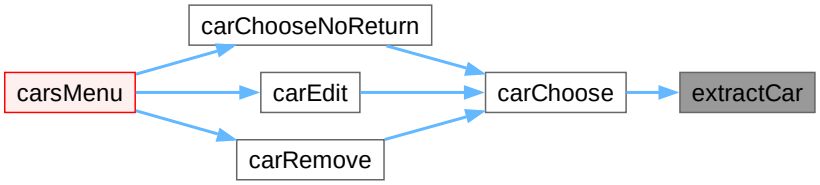
Parameters

out	Where to save the extracted Car.
node	From where the Car is extracted.

Definition at line 170 of file carsmenu.c.

Here is the call graph for this function:

Here is the caller graph for this function:



5.20 carsmenu.c

Go to the documentation of this file.

```
00001 #include "carsmenu.h"
00002 #include "car.h"
00003 #include "dbhandle.h"
00004 #include "list.h"
00005 #include "menuutil.h"
00006 #include <assert.h>
00007 #include <form.h>
00008 #include <ncurses.h>
00009 #include <panel.h>
00010 #include <stdio.h>
00011 #include <stdlib.h>
00012 #include <string.h>
00013
00014 #define NOTRACE
00015
00030 static bool carFormParse(struct Car **result, FORM *form) {
00031   assert(result && "Can not be null pointer.");
00032   bool isFormAltered = false;
00033   struct Car *resultPtr = *result;
00034
00035   // for each changed field in form
00036   for (int i = 0; i < field_count(form); ++i) {
00037     FIELD *curField = form_fields(form)[i];
00038     assert(curField);
00039     char *curFieldBuffer = field_buffer(curField, 0);
00040     assert(curFieldBuffer);
00041     // if field was edited
00042     if (field_status(curField) == true) {
00043       isFormAltered = true;
00044       // corresponding field indices in car structure.
00045       switch (i) {
00046       case 0:
00047         resultPtr->m_regNum = calloc(sizeof(char), strlen(curFieldBuffer) + 1);
00048         strcpy(resultPtr->m_regNum, curFieldBuffer);
00049         break;
00050       case 1:
00051         resultPtr->m_brand = calloc(sizeof(char), strlen(curFieldBuffer) + 1);
00052         strcpy(resultPtr->m_brand, curFieldBuffer);
00053         break;
00054       case 2:
00055         resultPtr->m_model = calloc(sizeof(char), strlen(curFieldBuffer) + 1);
00056         strcpy(resultPtr->m_model, curFieldBuffer);
00057         break;
00058       case 3:
00059         resultPtr->m_yOfProd = atoi(curFieldBuffer);
00060         break;
00061       case 4:
00062         resultPtr->m_color = calloc(sizeof(char), strlen(curFieldBuffer) + 1);
00063         strcpy(resultPtr->m_color, curFieldBuffer);
00064         break;
00065       case 5:
00066         resultPtr->m_mileage = atoi(curFieldBuffer);
00067         break;
00068       }
00069     }
00070   }
00071   return isFormAltered;
00072 }
00073
00083 static bool carFormEdit(struct Car **result,
00084                         const struct Car *const placeHolder) {
00085   assert(result);
00086   const char *const formFieldNames[] = {
00087       "Registration Number", "Brand", "Model",
00088       "Year of Production",  "Color", "Mileage (KM)"};
00089   int fieldCount = sizeof(formFieldNames) / sizeof(*formFieldNames);
00090
00091   FORM *form = formInit(fieldCount);
00092   set_field_type(form_fields(form)[3], TYPE_INTEGER, 0, 0, 0);
00093   set_field_type(form_fields(form)[5], TYPE_INTEGER, 0, 0, 0);
00094   if (placeHolder) {
00095     if (placeHolder->m_regNum) {
00096       set_field_buffer(form_fields(form)[0], 0, placeHolder->m_regNum);
00097     }
00098     if (placeHolder->m_brand) {
00099       set_field_buffer(form_fields(form)[1], 0, placeHolder->m_brand);
00100     }
00101     if (placeHolder->m_model) {
00102       set_field_buffer(form_fields(form)[2], 0, placeHolder->m_model);
00103     }
00104     if (placeHolder->m_yOfProd != INVALIDCARYOFPROD) {
00105       char *tempstr = calloc(FORMFIELDLENGTH, sizeof(char));
```

```
00106       sprintf(tempstr, "%d", placeHolder->m_yOfProd);
00107       set_field_buffer(form_fields(form)[3], 0, tempstr);
00108       free(tempstr);
00109     }
00110     if (placeHolder->m_color) {
00111       set_field_buffer(form_fields(form)[4], 0, placeHolder->m_color);
00112     }
00113     if (placeHolder->m_mileage != INVALIDCARMILEAGE) {
00114       char *tempstr = calloc(FORMFIELDLENGTH, sizeof(char));
00115       sprintf(tempstr, "%ld", placeHolder->m_mileage);
00116       set_field_buffer(form_fields(form)[5], 0, tempstr);
00117       free(tempstr);
00118     }
00119   }
00120   formInvoke(form, formFieldNames, "Car");
00121
00122   bool altered = false;
00123   altered = carFormParse(result, form);
00124
00125   formFree(form);
00126   return altered;
00127 }
00128
00132 void addCar(void) {
00133   struct Car *newCar = carNew();
00134   if (carFormEdit(&newCar, 0) && carIsComplete(newCar)) {
00135     if (!dbHandleCarInsert(newCar)) {
00136       const char *mess[] = {"Database error", NULL};
00137       menuUtilMessagebox("Adding car failed", (mess));
00138     }
00139   } else {
00140     const char *mess[] = {"Not all fields were set.", NULL};
00141     menuUtilMessagebox("Adding car failed", (mess));
00142   }
00143   carFree(newCar);
00144 }
00145
00152 char *carGetListViewString(const struct Car *car) {
00153   struct Car *carPtr = (struct Car *)car;
00154   // 6 is the number of fields in the resulting string
00155   const int fieldCount = 6;
00156   // +1 for null terminator
00157   char *result = calloc(fieldCount * FORMFIELDLENGTH + 1, sizeof(char));
00158   sprintf(result, "%*s%*s%*s%*d%*s%*ld", FORMFIELDLENGTH, carPtr->m_regNum,
00159           FORMFIELDLENGTH, carPtr->m_brand, FORMFIELDLENGTH, carPtr->m_model,
00160           FORMFIELDLENGTH, carPtr->m_yOfProd, FORMFIELDLENGTH, carPtr->m_color,
00161           FORMFIELDLENGTH, carPtr->m_mileage);
00162   return result;
00163 }
00164
00170 static void extractCar(struct Car **out, const struct ListNode *node) {
00171   assert(out != NULL && "extractCar cannot output to NULL");
00172   struct Car *res = node->m_data;
00173   carClone(out, node->m_data);
00174 }
00175
00185 static struct Car *carChoose(void) {
00186   const char *colNames[] = {"Registration Number", "Brand", "Model",
00187                             "Year of Production",  "Color", "Mileage (KM)"};
00188   const int colCount = sizeof(colNames) / sizeof(*colNames);
00189
00190   struct Car *out = NULL;
00191   bool didChoose = listViewInvoke(
00192       (void **)&out, (void *)(const struct ListNode *)extractCar, carGetList,
00193       colNames, colCount, (char *(*)(void *))carGetListViewString,
00194       (void *)(void *)carFree);
00195
00196 #ifndef NOTRACE
00197   if (out) {
00198     char *outVal = calloc(100, sizeof(char));
00199     char *msg[] = {carGetListViewString(out), NULL};
00200     sprintf(outVal, "carChoose -- out val = %p", out);
00201     menuUtilMessagebox(outVal, (const char **)msg);
00202     free(msg[0]);
00203     free(outVal);
00204   }
00205 #endif
00206
00207   doupdate();
00208   return out;
00209 }
00210
00214 void carRemove(void) {
00215   struct Car *toRemove = carChoose();
00216   if (toRemove) {
00217 #ifndef NOTRACE
00218     char *str = calloc(200, sizeof(char));
```

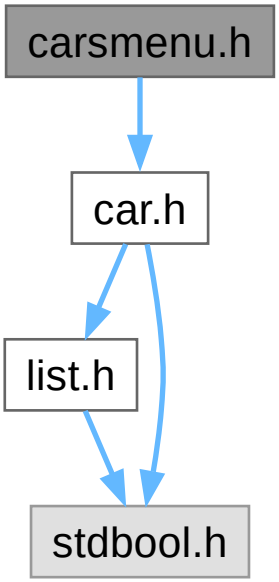
```
00219     char *info = carGetListViewString(toRemove);
00220     const char *msg[] = {"Removing car with data:", info, NULL};
00221     menuUtilMessagebox("carRemove", (const char **)msg);
00222 #endif
00223     dbHandleCarRemove(toRemove->m_ID);
00224     carFree(toRemove);
00225   }
00226 }
00227
00232 void carChooseNoReturn(void) {
00233   struct Car *r = carChoose();
00234   if (r)
00235     carFree(r);
00236 }
00237
00241 void carEdit(void) {
00242   struct Car *toEdit = carChoose();
00243   struct Car *edited = NULL;
00244   carClone(&edited, toEdit);
00245
00246   if (carFormEdit(&edited, toEdit)) {
00247     dbHandleCarUpdate(edited);
00248   }
00249
00250   carFree(toEdit);
00251   carFree(edited);
00252 }
00253
00257 void carsMenu(void) {
00258
00259   const char *const title = "Cars";
00260   const char *const choices[] = {"List cars", "Add car", "Remove car",
00261                                  "Edit car", "Return to main menu"};
00262   const int choicesCount = sizeof(choices) / sizeof(choices[0]);
00263   void (*menuFun[])(void) = {(void (*)(void))carChooseNoReturn, addCar,
00264                              carRemove, carEdit, NULL};
00265   menuInvoke(title, choices, choicesCount, menuFun);
00266 }
```

5.21 carsmenu.h File Reference

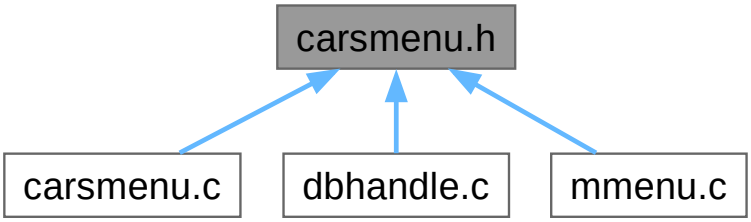
Cars menu interface.

```
#include ¨car.h¨
```

Include dependency graph for carsmenu.h:



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



Functions

- void carsMenu (void)

  Handles displaying of cars menu.
- char ∗ carGetListViewString (const struct Car ∗car)

  Given car, generates a string representing car data in listView friendly format (whole row).

5.21.1 Detailed Description

Cars menu interface.

Definition in file carsmenu.h.

5.21.2 Function Documentation

5.21.2.1 carGetListViewString()

```
char * carGetListViewString (
            const struct Car * car )
```

Given car, generates a string representing car data in listView friendly format (whole row).

Parameters

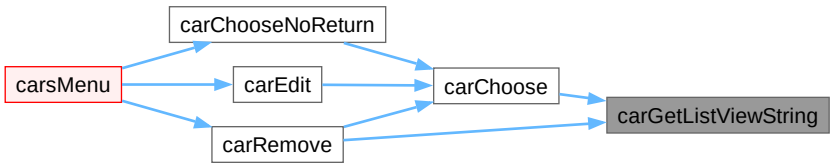
car	Car based on which to generate the string.

Returns

String representing the car.

Definition at line 152 of file carsmenu.c.

Here is the caller graph for this function:



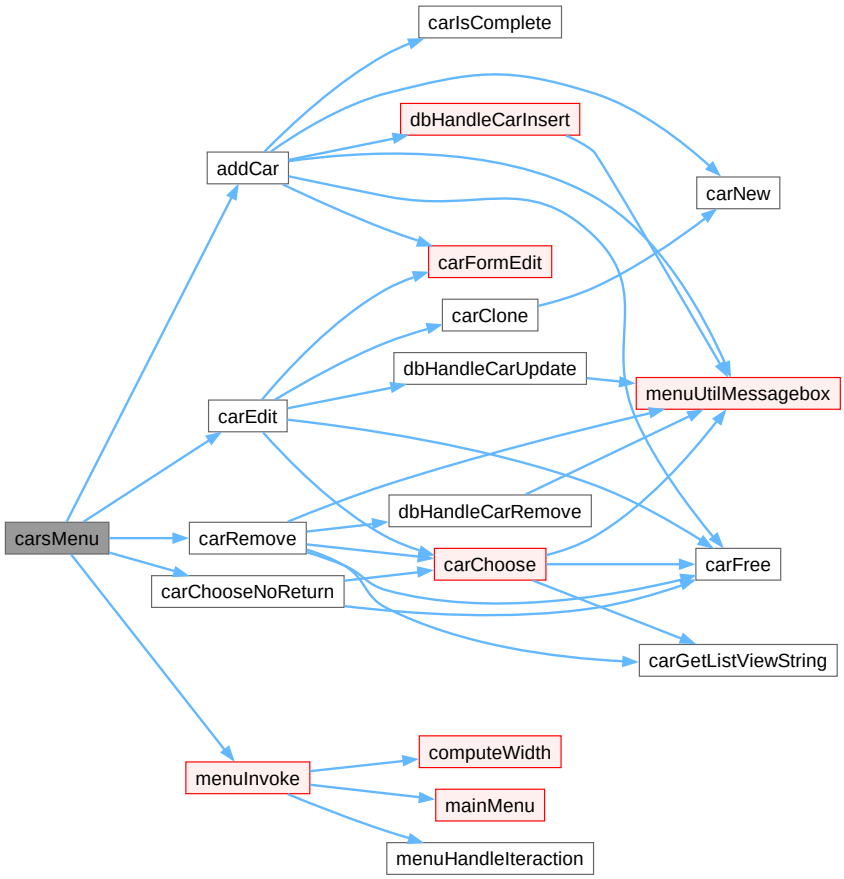
5.21.2.2 carsMenu()

```
void carsMenu (
            void  )
```

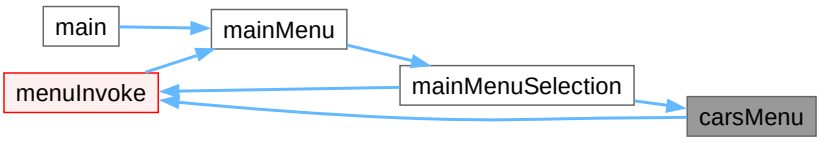
Handles displaying of cars menu.

Definition at line 257 of file carsmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.22 carsmenu.h

Go to the documentation of this file.

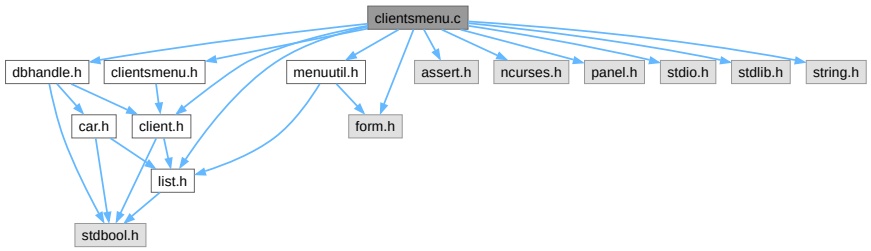
```
00001 #ifndef CARSMENU_H
00002 #define CARSMENU_H
00003
00010 #include "car.h"
00011
00012 void carsMenu(void);
00013
00014 char *carGetListViewString(const struct Car *car);
00015 #endif // CARSMENU_H
```

5.23 clientsmenu.c File Reference

Clients menu implementation.

```
#include ¨clientsmenu.h¨
#include ¨client.h¨
#include ¨dbhandle.h¨
#include ¨list.h¨
#include ¨menuutil.h¨
#include <assert.h>
#include <form.h>
#include <ncurses.h>
#include <panel.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for clientsmenu.c:



Functions

- static bool clientFormParse (struct Client ∗∗result, FORM ∗form)

  Parse form.
- static bool clientFormEdit (struct Client ∗∗result, const struct Client ∗const placeHolder)

  Function for editing client.
- void addClient (void)

  function for adding client;
- char ∗ clientGetListViewString (const struct Client ∗client)

  Given client generates string repesenting client string in listView friendly format. (whole row)
- static void extractClient (struct Client ∗∗out, const struct ListNode ∗node)

  extract client given ListNode.
- static struct Client ∗ clientChoose (void)

  Invokes ListView of clients.
- void clientRemove (void)

  Remove client.
- void clientChooseNoReturn (void)

  Wrapper around clientChoose frees extracted client instanntly.
- void clientEdit (void)

  Edit client.
- void clientsMenu (void)

  Handles displaying of clients menu.

5.23.1 Detailed Description

Clients menu implementation.

Definition in file clientsmenu.c.

5.23.2 Macro Definition Documentation

5.23.2.1 NOTRACE

```
#define NOTRACE
```

Definition at line 14 of file clientsmenu.c.

5.23.3 Function Documentation

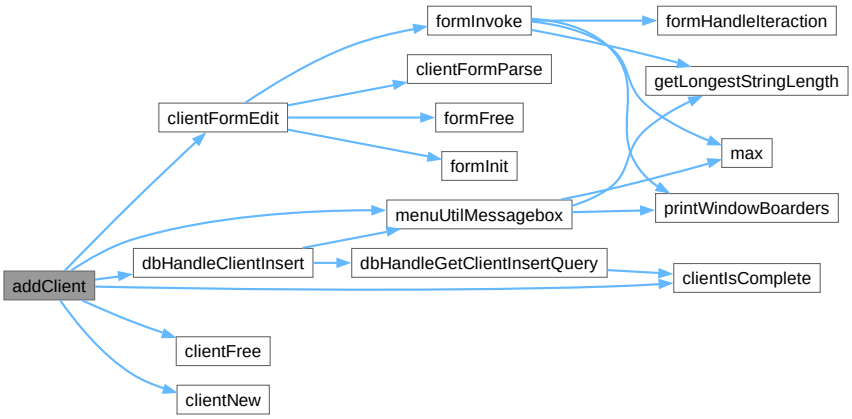
5.23.3.1 addClient()

```
void addClient (
            void  )
```

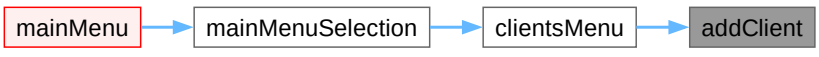
function for adding client;

Definition at line 125 of file clientsmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.23.3.2 clientChoose()

```
static struct Client * clientChoose (
            void  )  [static]
```

Invokes ListView of clients.

Returns

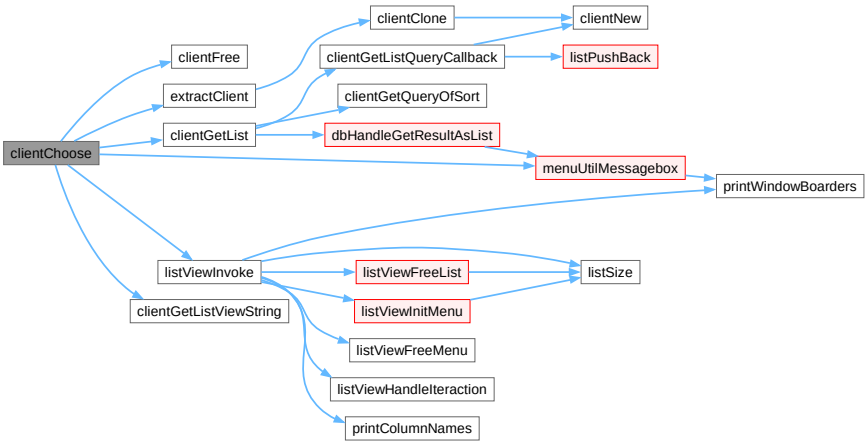
-Chosen client clone -NULL if canceled.

Note

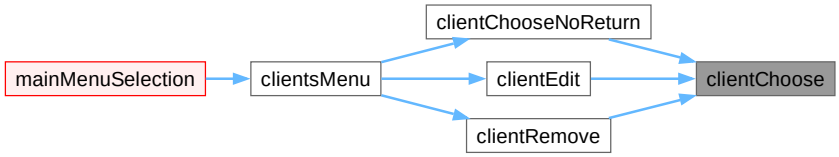
Extracted client has to be freed manually. See also clientChooseNoReturn.

Definition at line 178 of file clientsmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:



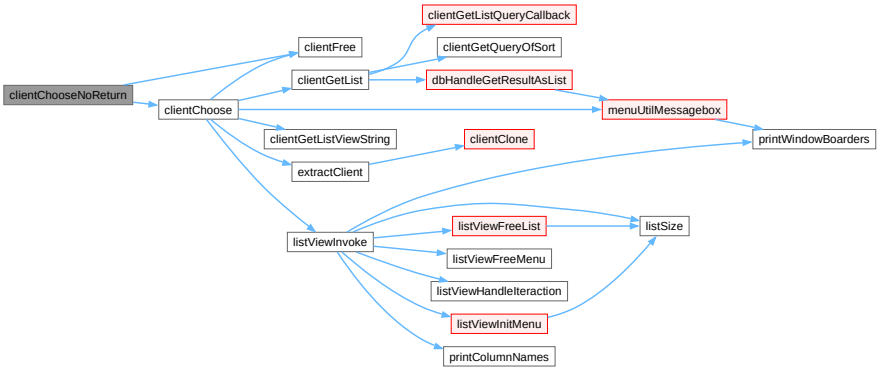
5.23.3.3 clientChooseNoReturn()

```
void clientChooseNoReturn (
            void  )
```

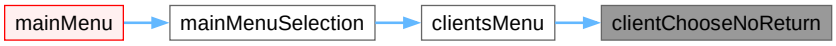
Wrapper around clientChoose frees extracted client instanntly.

Definition at line 225 of file clientsmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:



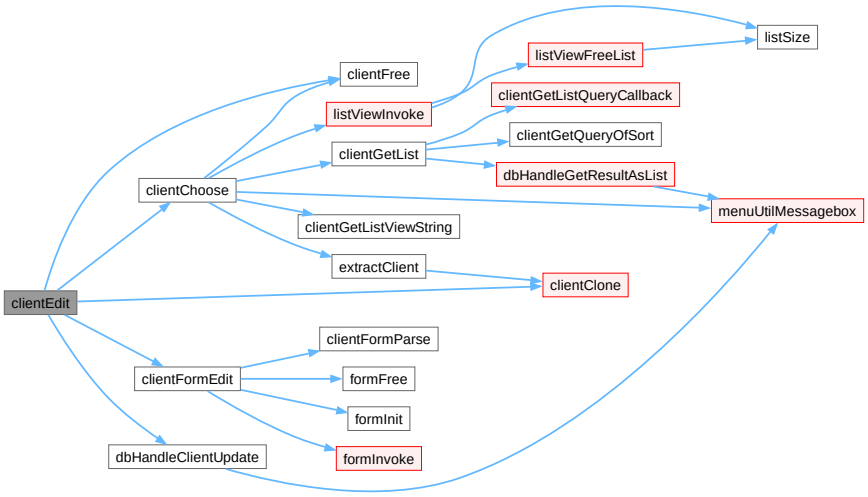
5.23.3.4 clientEdit()

```
void clientEdit (
            void  )
```

Edit client.

Definition at line 234 of file clientsmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:

5.23.3.5 clientFormEdit()

```
static bool clientFormEdit (
            struct Client ** result,
            const struct Client *const placeHolder )  [static]
```

Function for editing client.

Parameters

result	Where to put results at.
placeHolder	Structure of placeholder values for form. If NULL placeholders are empty (usefull when instead of editing adding).

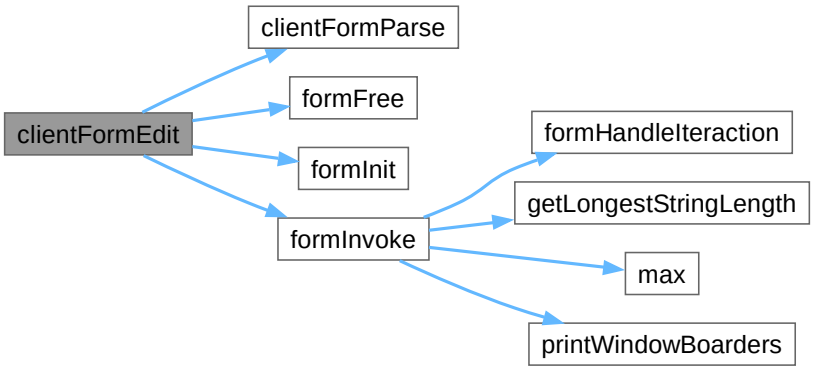
Returns

Whenever changes should be propagated.

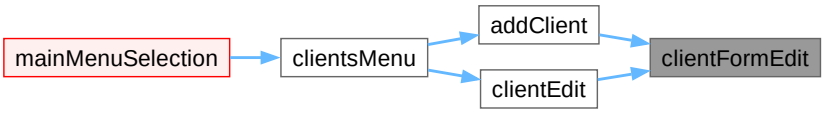
- false = nothing to do.

Definition at line 80 of file clientsmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.23.3.6 clientFormParse()

```
static bool clientFormParse (
            struct Client ** result,
            FORM * form )  [static]
```

Parse form.

Parameters

result	Client object where to save parsed result, Client::m_ID will be set to INVALIDCLIENTID. Does not allocate object.
form	Filled already form containing client data.

Returns

- True if any of fields has been altered
- False if none of fields has been altered.

Definition at line 31 of file clientsmenu.c.

Here is the caller graph for this function:

5.23.3.7 clientGetListViewString()

```
char * clientGetListViewString (
            const struct Client * client )
```

Given client generates string repesenting client string in listView friendly format. (whole row)

Parameters

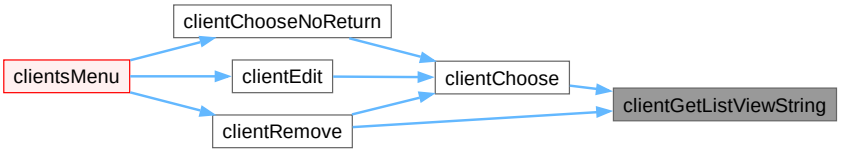
client	Client based on which generate string.

Returns

string repersenting client

Definition at line 145 of file clientsmenu.c.

Here is the caller graph for this function:



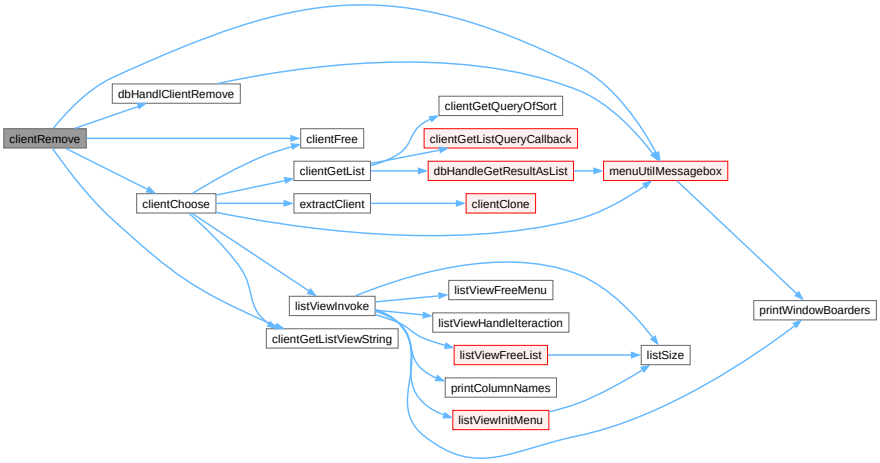
5.23.3.8 clientRemove()

```
void clientRemove (
            void  )
```

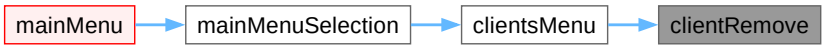
Remove client.

Definition at line 207 of file clientsmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.23.3.9 clientsMenu()

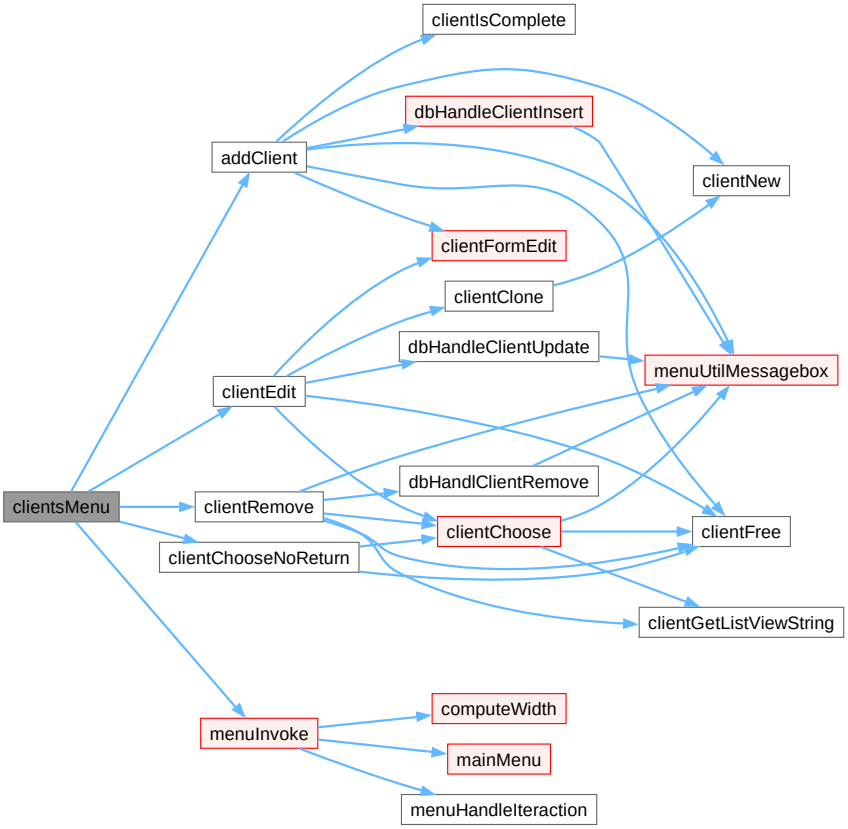
```
void clientsMenu (
            void  )
```

Handles displaying of clients menu.

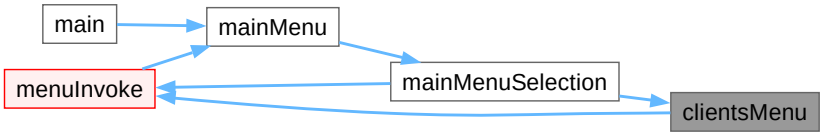
Todo implement submenus.

Definition at line 250 of file clientsmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.23.3.10 extractClient()

```
static void extractClient (
            struct Client ** out,
            const struct ListNode * node )  [static]
```

extract client given ListNode.

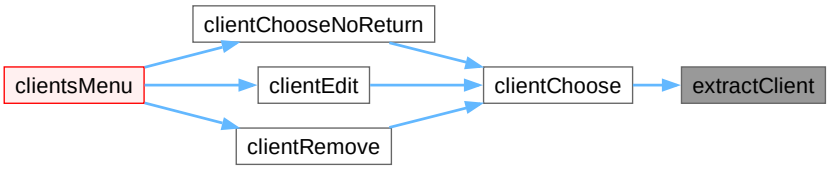
Parameters

out	where to save extracted Client.
node	from where Client is extracted.

Definition at line 163 of file clientsmenu.c.

Here is the call graph for this function:

Here is the caller graph for this function:



5.24 clientsmenu.c

Go to the documentation of this file.

```
00001 #include "clientsmenu.h"
00002 #include "client.h"
00003 #include "dbhandle.h"
00004 #include "list.h"
00005 #include "menuutil.h"
00006 #include <assert.h>
00007 #include <form.h>
00008 #include <ncurses.h>
00009 #include <panel.h>
00010 #include <stdio.h>
00011 #include <stdlib.h>
00012 #include <string.h>
00013
00014 #define NOTRACE
00015
00031 static bool clientFormParse(struct Client **result, FORM *form) {
00032   assert(result && "Can not be null pointer.");
00033   bool isFormAltered = false;
00034   struct Client *resultPtr = *result;
00035
00036   // for each changed field in form
00037   for (int i = 0; i < field_count(form); ++i) {
00038     FIELD *curField = form_fields(form)[i];
00039     assert(curField);
00040     char *curFieldBuffer = field_buffer(curField, 0);
00041     assert(curFieldBuffer);
00042     // if field was edited
00043     if (field_status(curField) == true) {
00044       isFormAltered = true;
00045       // corresponding field indices in  client structure.
00046       switch (i) {
00047       case 0:
00048         resultPtr->m_cardID = atoi(curFieldBuffer);
00049         break;
00050       case 1:
00051         resultPtr->m_name = calloc(sizeof(char), strlen(curFieldBuffer) + 1);
00052         strcpy(resultPtr->m_name, curFieldBuffer);
00053         break;
00054       case 2:
```

```
00055         resultPtr->m_surname = calloc(sizeof(char), strlen(curFieldBuffer) + 1);
00056         strcpy(resultPtr->m_surname, curFieldBuffer);
00057         break;
00058       case 3:
00059         resultPtr->m_adress = calloc(sizeof(char), strlen(curFieldBuffer) + 1);
00060         strcpy(resultPtr->m_adress, curFieldBuffer);
00061         break;
00062       case 4:
00063         resultPtr->m_phoneNum = atoi(curFieldBuffer);
00064         break;
00065       }
00066     }
00067   }
00068   return isFormAltered;
00069 }
00070
00080 static bool clientFormEdit(struct Client **result,
00081                            const struct Client *const placeHolder) {
00082   assert(result);
00083   const char *const formFieldNames[] = {"Card id", "Name", "Surname", "Address",
00084                                         "Phone Number"};
00085   int fieldCount = sizeof(formFieldNames) / sizeof(*formFieldNames);
00086
00087   FORM *form = formInit(fieldCount);
00088   set_field_type(form_fields(form)[0], TYPE_INTEGER, 0, 0, 0);
00089   set_field_type(form_fields(form)[4], TYPE_INTEGER, 0, 0, 0);
00090   if (placeHolder) {
00091     if (placeHolder->m_cardID != INVALIDCLIENTCARDID) {
00092       char *tempstr = calloc(FORMFIELDLENGTH, sizeof(char));
00093       sprintf(tempstr, "%d", placeHolder->m_cardID);
00094       set_field_buffer(form_fields(form)[0], 0, tempstr);
00095       free(tempstr);
00096     }
00097     if (placeHolder->m_name) {
00098       set_field_buffer(form_fields(form)[1], 0, placeHolder->m_name);
00099     }
00100     if (placeHolder->m_surname) {
00101       set_field_buffer(form_fields(form)[2], 0, placeHolder->m_surname);
00102     }
00103     if (placeHolder->m_adress) {
00104       set_field_buffer(form_fields(form)[3], 0, placeHolder->m_adress);
00105     }
00106     if (placeHolder->m_phoneNum != INVALIDCLIENTPHONENUM) {
00107       char *tempstr = calloc(FORMFIELDLENGTH, sizeof(char));
00108       sprintf(tempstr, "%d", placeHolder->m_phoneNum);
00109       set_field_buffer(form_fields(form)[4], 0, tempstr);
00110       free(tempstr);
00111     }
00112   }
00113   formInvoke(form, formFieldNames, "Client");
00114
00115   bool altered = false;
00116   altered = clientFormParse(result, form);
00117
00118   formFree(form);
00119   return altered;
00120 }
00121
00125 void addClient(void) {
00126   struct Client *newClient = clientNew();
00127   if (clientFormEdit(&newClient, 0) && clientIsComplete(newClient)) {
00128     if (!dbHandleClientInsert(newClient)) {
00129       const char *mess[] = {"Database error", NULL};
00130       menuUtilMessagebox("Adding client failed", (mess));
00131     }
00132   } else {
00133     const char *mess[] = {"Not all fields were set.", NULL};
00134     menuUtilMessagebox("Adding client failed", (mess));
00135   }
00136   clientFree(newClient);
00137 }
00138
00145 char *clientGetListViewString(const struct Client *client) {
00146   struct Client *clientPtr = (struct Client *)client;
00147   // 5 is number of fields in resulting string
00148   const int fieldCount = 5;
00149   // +1 for null termintaor
00150   char *result = calloc(fieldCount * FORMFIELDLENGTH + 1, sizeof(char));
00151   sprintf(result, "%*d%*s%*s%*s%*d", FORMFIELDLENGTH, clientPtr->m_cardID,
00152           FORMFIELDLENGTH, clientPtr->m_name, FORMFIELDLENGTH,
00153           clientPtr->m_surname, FORMFIELDLENGTH, clientPtr->m_adress,
00154           FORMFIELDLENGTH, clientPtr->m_phoneNum);
00155   return result;
00156 }
00157
00163 static void extractClient(struct Client **out, const struct ListNode *node) {
00164   assert(out != NULL && "extractClient can not output to NULL");
```

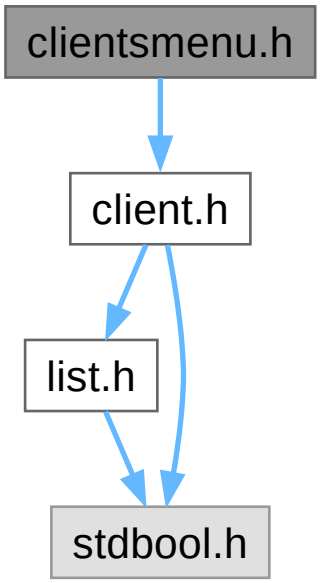
```
00165   struct Client *res = node->m_data;
00166   clientClone(out, node->m_data);
00167 }
00168
00178 static struct Client *clientChoose(void) {
00179   const char *colNames[] = {"cardId", "name", "surname", "adress",
00180                             "phone number"};
00181   const int colCount = sizeof(colNames) / sizeof(*colNames);
00182
00183   struct Client *out = NULL;
00184   bool didChoose = listViewInvoke(
00185       (void **)&out, (void *)(const struct ListNode *)extractClient,
00186       clientGetList, colNames, colCount,
00187       (char *(*)(void *))clientGetListViewString, (void *)(void *)clientFree);
00188
00189 #ifndef NOTRACE
00190   if (out) {
00191     char *outVal = calloc(100, sizeof(char));
00192     char *msg[] = {clientGetListViewString(out), NULL};
00193     sprintf(outVal, "clientChoose -- out val = %p", out);
00194     menuUtilMessagebox(outVal, (const char **)msg);
00195     free(msg[0]);
00196     free(outVal);
00197   }
00198 #endif
00199
00200   doupdate();
00201   return out;
00202 }
00203
00207 void clientRemove(void) {
00208   struct Client *toRemove = clientChoose();
00209   if (toRemove) {
00210 #ifndef NOTRACE
00211     char *str = calloc(200, sizeof(char));
00212     char *info = clientGetListViewString(toRemove);
00213     const char *msg[] = {"Removing clinet with data:", info, NULL};
00214     menuUtilMessagebox("clientRemove", (const char **)msg);
00215 #endif
00216     dbHandlClientRemove(toRemove->m_ID);
00217     clientFree(toRemove);
00218   }
00219 }
00220
00225 void clientChooseNoReturn(void) {
00226   struct Client *r = clientChoose();
00227   if (r)
00228     clientFree(r);
00229 }
00230
00234 void clientEdit(void) {
00235   struct Client *toEdit = clientChoose();
00236   struct Client *edited = NULL;
00237   clientClone(&edited, toEdit);
00238
00239   if (clientFormEdit(&edited, toEdit)) {
00240     dbHandleClientUpdate(edited);
00241   }
00242
00243   clientFree(toEdit);
00244   clientFree(edited);
00245 }
00246
00250 void clientsMenu(void) {
00251
00252   const char *const title = "Clients";
00253   const char *const choices[] = {"List clients", "Add client", "Remove client",
00254                                  "Edit clients", "Return to main menu"};
00255   const int choicesCount = sizeof(choices) / sizeof(choices[0]);
00257   void (*menuFun[])(void) = {(void (*)(void))clientChooseNoReturn, addClient,
00258                              clientRemove, clientEdit, NULL};
00259   menuInvoke(title, choices, choicesCount, menuFun);
00260 }
```

5.25 clientsmenu.h File Reference

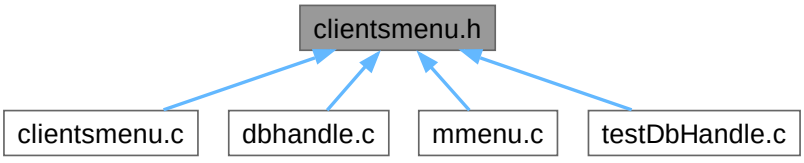
Clients menu interface.

```
#include ¨client.h¨
```

Include dependency graph for clientsmenu.h:



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



Functions

- void clientsMenu (void)

  Handles displaying of clients menu.
- char ∗ clientGetListViewString (const struct Client ∗client)

  Given client generates string repesenting client string in listView friendly format. (whole row)

5.25.1 Detailed Description

Clients menu interface.

Definition in file clientsmenu.h.

5.25.2 Function Documentation

5.25.2.1 clientGetListViewString()

```
char * clientGetListViewString (
            const struct Client * client )
```

Given client generates string repesenting client string in listView friendly format. (whole row)

Parameters

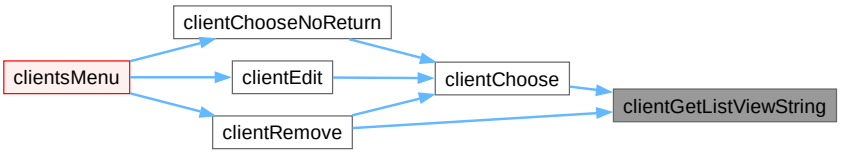
client	Client based on which generate string.

Returns

string repersenting client

Definition at line 145 of file clientsmenu.c.

Here is the caller graph for this function:



5.25.2.2 clientsMenu()

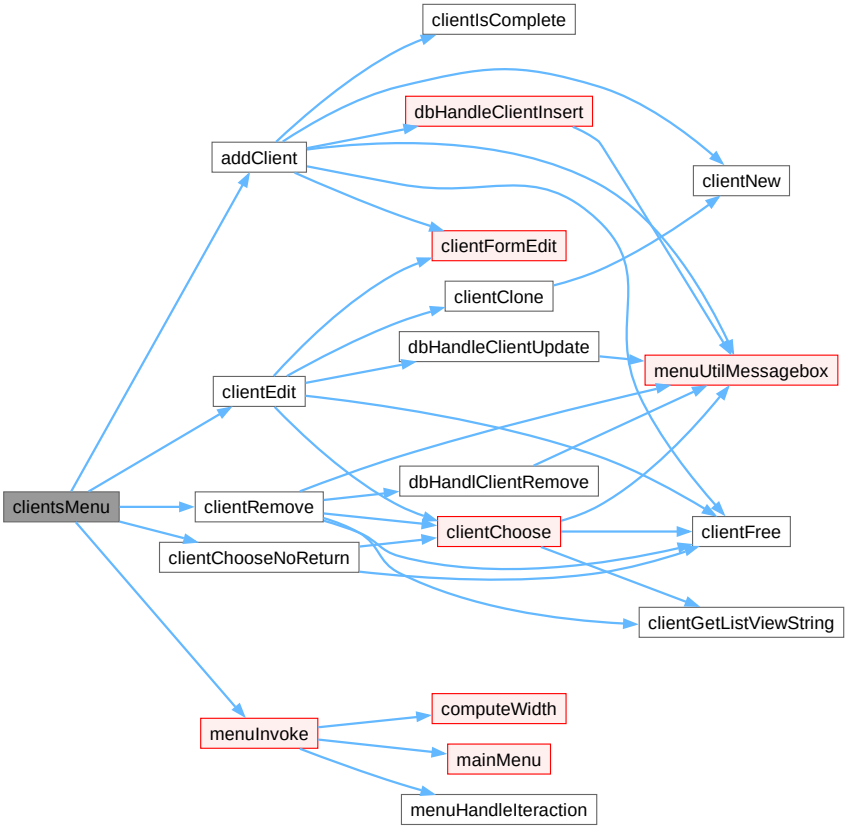
```
void clientsMenu (
            void  )
```

Handles displaying of clients menu.

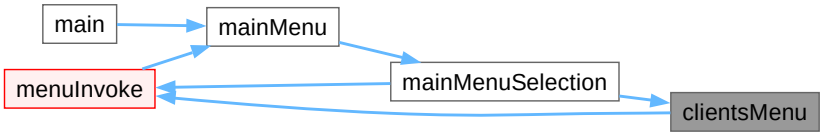
Todo implement submenus.

Definition at line 250 of file clientsmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.26 clientsmenu.h

Go to the documentation of this file.

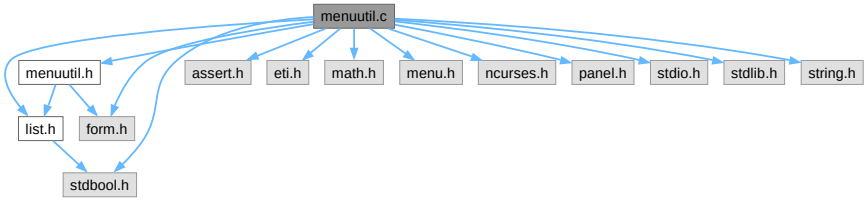
```
00001 #ifndef CLEINTSMENU_H
00002 #define CLEINTSMENU_H
00003
00009 #include "client.h"
00010
00011 void clientsMenu(void);
00012
00013 char *clientGetListViewString(const struct Client *client);
00014 #endif // CLEINTSMENU_H
```

5.27 menuutil.c File Reference

Menu displaying utilities.

```
#include ¨menuutil.h¨
#include ¨list.h¨
#include <assert.h>
#include <eti.h>
#include <form.h>
#include <math.h>
#include <menu.h>
#include <ncurses.h>
#include <panel.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for menuutil.c:



Macros

- #define NOTRACE

  Hide trace information used for debugging.
- #define MENUMARK (¨ ∗ ¨)

  String to indicate current selected choice in menus.

Enumerations

- enum ListViewIteractionStateCode {
  sortNext , sortPrev , sortInvert , canceled ,
  chosen , invalid }

  Status code returned by listViewHandleIteraction.

Functions

- int max (const int a, const int b)

  Add basic functionality to the shitty language which C is.
- int getLongestStringLength (const char ∗const stringArr[ ], const int stringsCount)

  Get length of longest string in array of strings.
- int computeWidth (const char ∗const title, const char ∗const choices[ ], const int optionsCount)

  Calculate minimum window width.
- void printWindowBoarders (WINDOW ∗window, const char ∗const title)

Print boarders of window.

- void menuUtilMessagebox (const char ∗const title, const char ∗const message[ ])

  Displays message box on the screen with title and message.
- static void menuHandleIteraction (MENU ∗menu, PANEL ∗panel)

  Control menu navigation and invoke option that we chose.
- void menuInvoke (const char ∗const title, const char ∗const choices[ ], const int choicesCount, void(∗menu↩ Fun[ ])(void))

  Handle all operation and functions for menu.
- static void formHandleIteraction (FORM ∗form)

  Manages input in the form.
- void formInvoke (FORM ∗form, const char ∗const formFieldNames[ ], const char ∗const title)

  Put form on screen in nice looking form.
- FORM ∗ formInit (const int fieldCount)

  Initialize FORM.
- void formFree (FORM ∗form)

  Frees memory used for form, and it's fields.
- static enum ListViewIteractionStateCode listViewHandleIteraction (struct ListNode ∗∗result, MENU ∗menu)
- static void listViewFreeList (struct List ∗∗list, void(∗dealloactor)(void ∗))

  frees internal List of listView.
- static MENU ∗ listViewInitMenu (struct List ∗list, char ∗(∗getItemString)(void ∗), const int colCount)

  Allocates and fills MENU, based on List content.
- static void listViewFreeMenu (MENU ∗menu, const int itemCount)

  frees memory used by listView internal Menu
- void printColumnNames (WINDOW ∗win, const char ∗const columnNames[ ], const int colCount, const int current)

  Prints header row of table.
- bool listViewInvoke (void ∗∗out, void(∗dataExtractor)(void ∗∗out, const struct ListNode ∗const data), struct List ∗(∗listFun)(int sortType, bool descending), const char ∗const columnNames[ ], const int colCount, char ∗(∗getItemString)(void ∗), void(∗dealloactor)(void ∗))

  List Viewer for lists.

5.27.1 Detailed Description

Menu displaying utilities.

Wraps ncurses liblary.

Definition in file menuutil.c.

5.27.2 Macro Definition Documentation

5.27.2.1 MENUMARK

```
#define MENUMARK (¨ ∗ ¨)
```

String to indicate current selected choice in menus.

Definition at line 29 of file menuutil.c.

5.27.2.2 NOTRACE

```
#define NOTRACE
```

Hide trace information used for debugging.

Definition at line 24 of file menuutil.c.

5.27.3 Enumeration Type Documentation

5.27.3.1 ListViewIteractionStateCode

```
enum ListViewIteractionStateCode
```

Status code returned by listViewHandleIteraction.

Enumerator

sortNext	Requestesed next sorting type.
sortPrev	Requestesed previous soring type.
sortInvert	Requestesed invertion of sort ASC/DESC.
canceled	Canceled, no MENU ITEM was chosen.
chosen	Chosen MENU ITEM.
invalid	Invalid state should never happen.

Definition at line 383 of file menuutil.c.

5.27.4 Function Documentation

5.27.4.1 computeWidth()

```
int computeWidth (
            const char *const title,
            const char *const choices[],
            const int optionsCount )
```

Calculate minimum window width.

Parameters

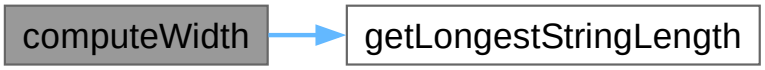
title	Char pointer to title.
choices	Char pointer to table of choices
optionsCount	Number of elements in table of choices

Returns

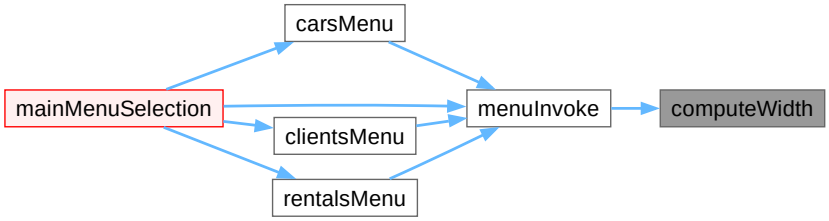
Minimum number of columns needed

Definition at line 64 of file menuutil.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.27.4.2 formFree()

```
void formFree (
            FORM * form )
```

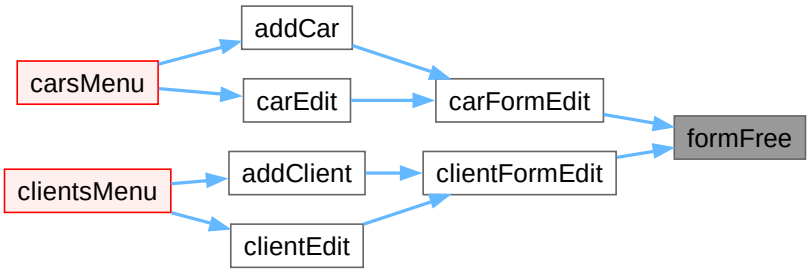
Frees memory used for form, and it's fields.

Parameters

form	Form to be free.

Definition at line 370 of file menuutil.c.

Here is the caller graph for this function:



5.27.4.3 formHandleIteraction()

```
static void formHandleIteraction (
            FORM * form )  [static]
```

Manages input in the form.

Parameters

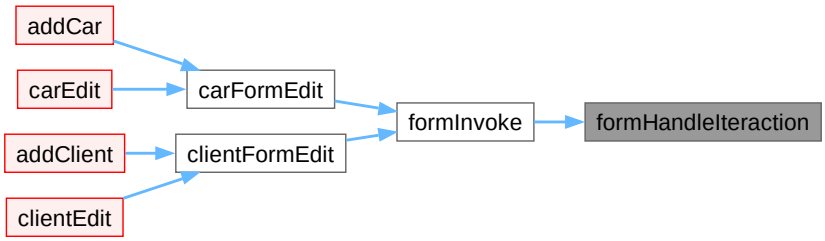
form	form that input will go into .

Todo Resign form form, change return type.

Todo Display information about invalid data in current field.

Definition at line 229 of file menuutil.c.

Here is the caller graph for this function:



5.27.4.4 formInit()

```
FORM * formInit (
            const int fieldCount )
```

Initialize FORM.

Parameters

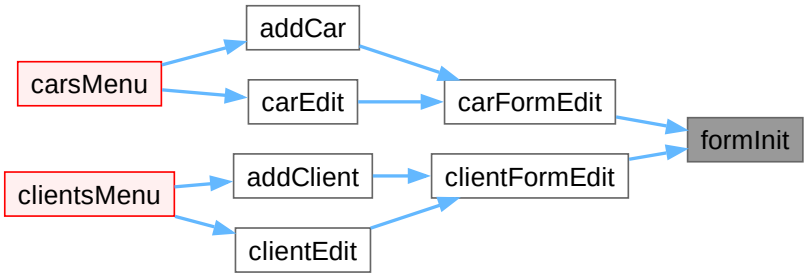
fieldCount	How many fields will be in the field.

Returns

initialized FORM pointer.

Definition at line 351 of file menuutil.c.

Here is the caller graph for this function:



5.27.4.5 formInvoke()

```
void formInvoke (
            FORM * form,
            const char *const formFieldNames[],
            const char *const title )
```

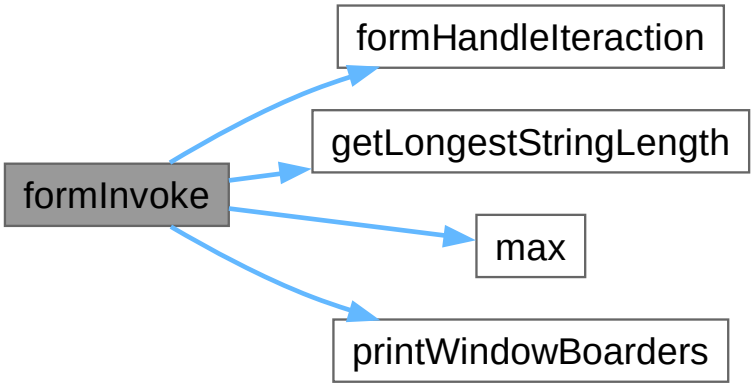
Put form on screen in nice looking form.

Parameters

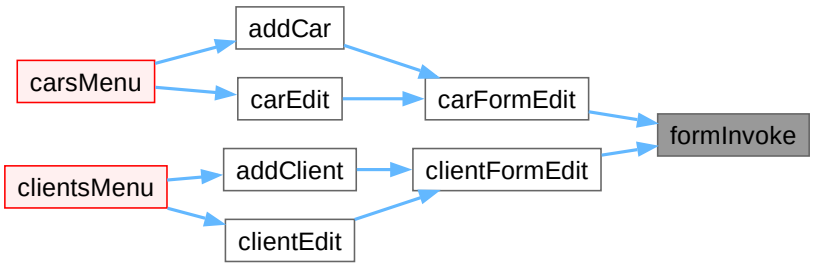
form	Form that will be put on scree.
formFieldNames	array of field names.
title	Title of window(form)

Definition at line 289 of file menuutil.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.27.4.6 getLongestStringLength()

```
int getLongestStringLength (
            const char *const stringArr[],
            const int stringsCount )
```

Get length of longest string in array of strings.

Parameters

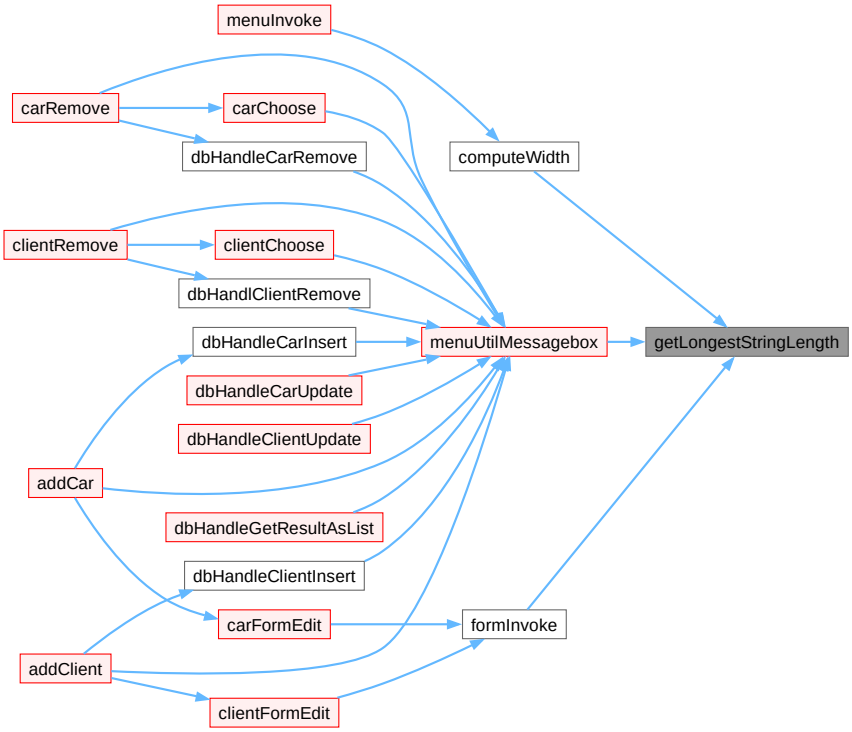
stringArr	Array of strings to look for longest string at.
stringsCount	How many strings are in the array.

Returns

Length of longest string in array.

Definition at line 45 of file menuutil.c.

Here is the caller graph for this function:



5.27.4.7 listViewFreeList()

```
static void listViewFreeList (
            struct List ** list,
            void(*)(void *) dealloactor )  [static]
```

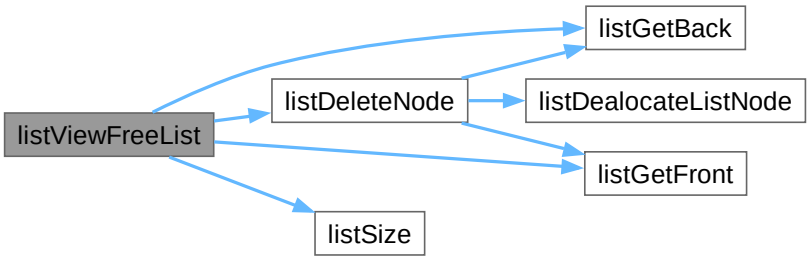
frees internal List of listView.

Parameters

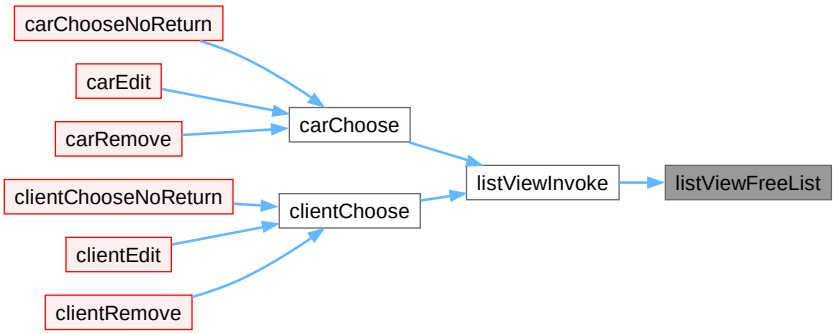
list	Internal List of ListView.
dealloactor	Function deallocating data from the list.

Definition at line 459 of file menuutil.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.27.4.8 listViewFreeMenu()

```
static void listViewFreeMenu (
            MENU * menu,
            const int itemCount )  [static]
```

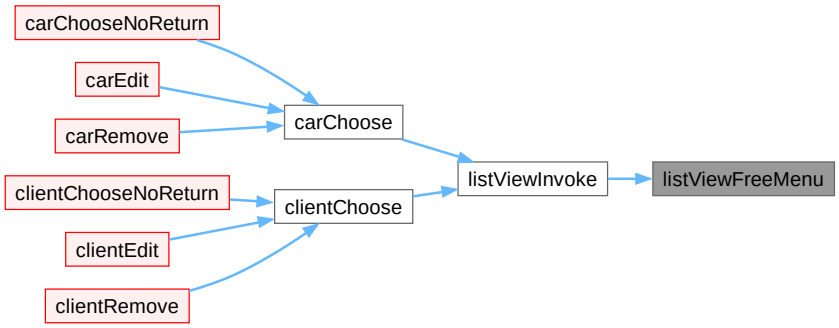
frees memory used by listView internal Menu

Parameters

menu	Menu to be freed.
itemCount	Count of items in menu.

Definition at line 524 of file menuutil.c.

Here is the caller graph for this function:



5.27.4.9 listViewHandleIteraction()

```
static enum ListViewIteractionStateCode listViewHandleIteraction (
            struct ListNode ** result,
            MENU * menu )  [static]
```

Parameters

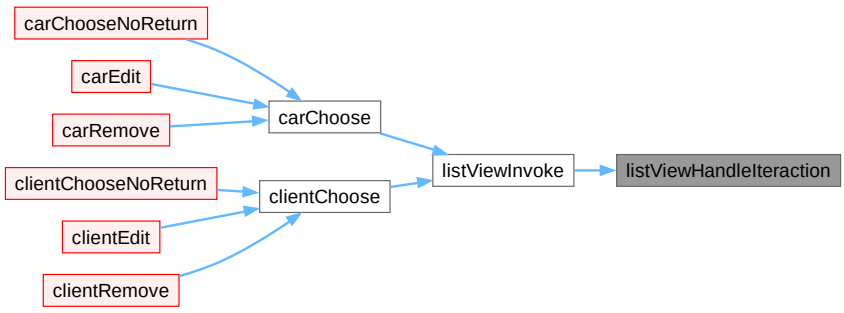
result	usr_ptr of chosen menu ITEM will be assigned to this.
menu	Menu to choose from.

Returns

ListViewIteractionStateCode

Definition at line 404 of file menuutil.c.

Here is the caller graph for this function:



5.27.4.10 listViewInitMenu()

```
static MENU * listViewInitMenu (
             struct List * list,
             char *(*)(void *) getItemString,
             const int colCount )  [static]
```

Allocates and fills MENU, based on List content.

Parameters

list	List based on which MENU will be created.
getItemString	Function creating string based on ListNode::m_data(it's passed as praemeter). Should only set ITEM name and description.
colCount	how many coulumns are to be dsiplayed.

Returns

menu based on list.

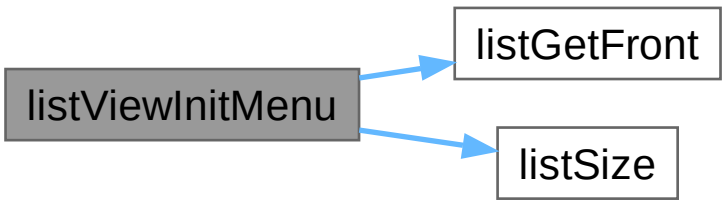
Every ITEM usr_pointer points to ListNode which it repersents.

Warning

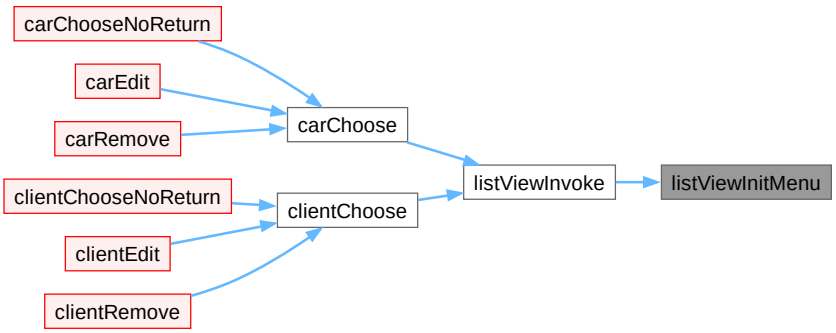
Does not use weaper around ListNode to get ListNode::m_data that is passsed to getItem function as parameter.

Definition at line 485 of file menuutil.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.27.4.11 listViewInvoke()

```
bool listViewInvoke (
            void ** out,
            void(*)(void **out, const struct ListNode *const data) dataExtractor,
            struct List *(*)(int sortType, bool descending) listFun,
            const char *const columnNames[],
            const int colCount,
            char *(*)(void *) getItemString,
            void(*)(void *) dealloactor )
```

List Viewer for lists.

Parameters

out	Where result will be saved.
dataExtractor	Function taking two parameters first is pointer to the memory where result will be saved (out parameter will be passed internally), second is Listnode, from witch data will be extracted.

Note

dataExtractor parameter function receives pointer to internal listViewInvoke List ListNode that is deallocated after call returns so if result is to be preserved it has to do copy of data by itself.

Parameters

listFun	Functions that returns sorted list. Parameter sortType says which sort type should be used it be handled by function. Parameter descending of informs if sorting is ascending or descending.

Note

listFuns sortType paremeter should be handled in range from 0 to colCount-1.

Parameters

columnNames	array of column names strings.
colCount	How many columns are there.
getItemString	Function creating string based on ListNode::m_data(it's passed as praemeter). Should do padding.
dealloactor	Function deallocating data that is held in ListNode, not ListNode itself. Required for internal List dealocation.

Returns

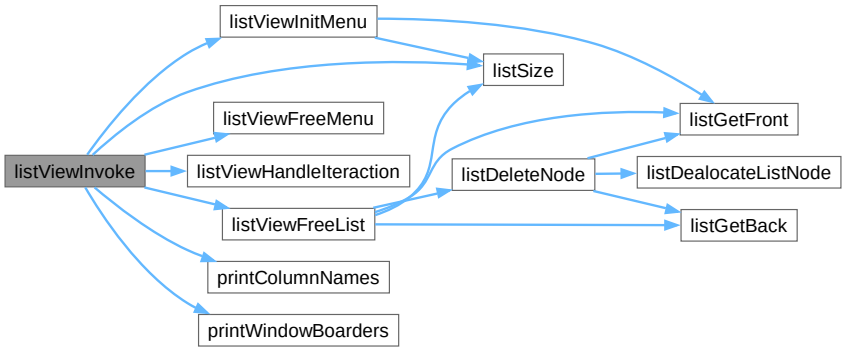
- true if chosen something.
- false if canceled.

Warning

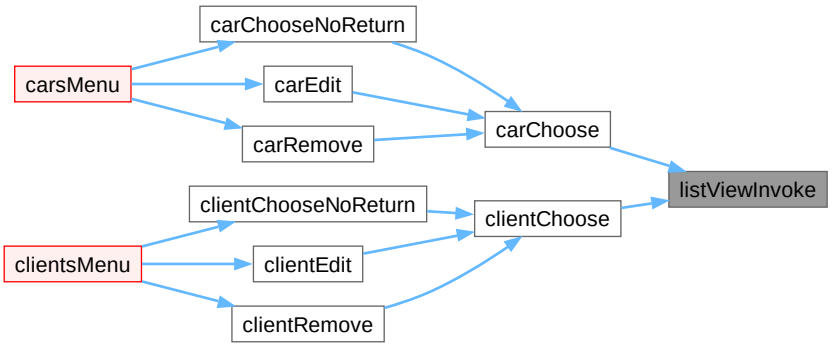
make sure extractData correctly, hard to debug.

Definition at line 583 of file menuutil.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.27.4.12 max()

```
int max (
            const int a,
            const int b )
```

Add basic functionality to the shitty language which C is.

Parameters

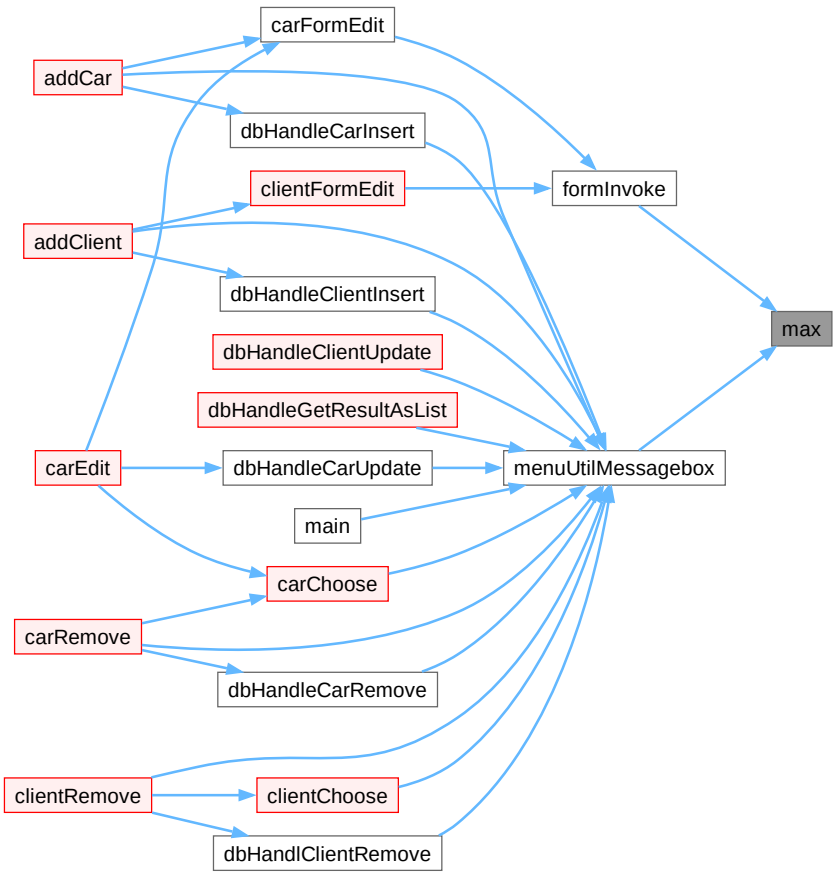
a	First value to compare.
b	Second value to compare.

Returns

Maximum of parameters passed.

Definition at line 37 of file menuutil.c.

Here is the caller graph for this function:



5.27.4.13 menuHandleIteraction()

```
static void menuHandleIteraction (
            MENU * menu,
            PANEL * panel )  [static]
```

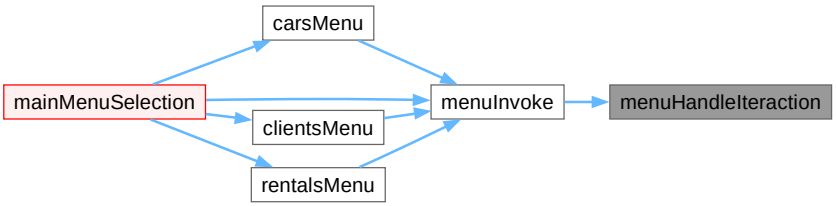
Control menu navigation and invoke option that we chose.

Parameters

menu	MENU pointer
panel	PANEL pointer

Definition at line 132 of file menuutil.c.

Here is the caller graph for this function:



5.27.4.14 menuInvoke()

```
void menuInvoke (
            const char *const title,
            const char *const choices[],
            const int choicesCount,
            void(*[])(void) menuFun )
```

Handle all operation and functions for menu.

amount of columns of the form field.

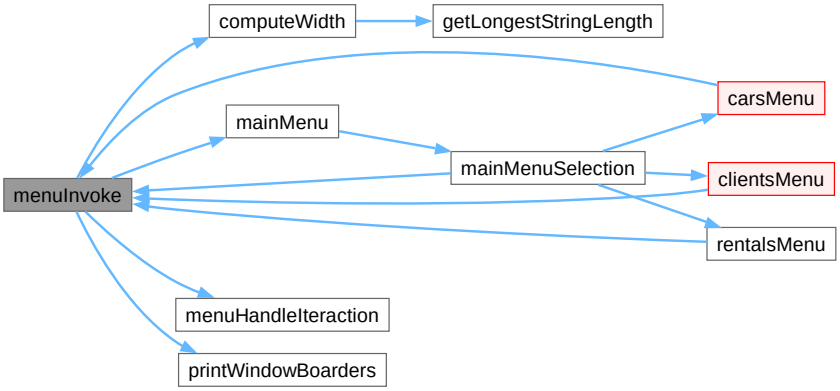
Parameters

title	Char pointer to title of menu
choices	Char pointer to table of choices
choicesCount	Number of elements in table of choices
menuFun	Table of pointers on functions

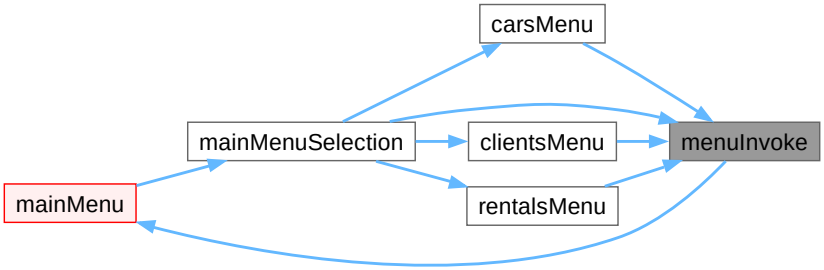
Todo Split into functions, make allocation and dallcation seperate functions, make it allocate on heap instead of stack.

Definition at line 179 of file menuutil.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.27.4.15 menuUtilMessagebox()

```
void menuUtilMessagebox (
            const char *const title,
            const char *const message[] )
```

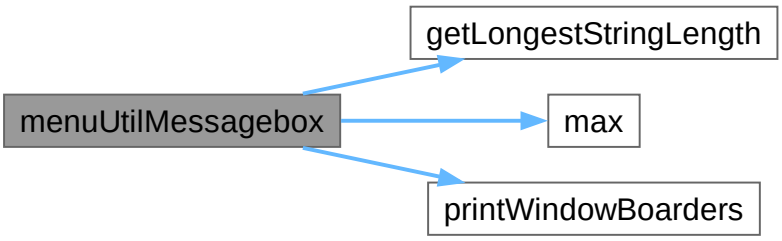
Displays message box on the screen with title and message.

Parameters

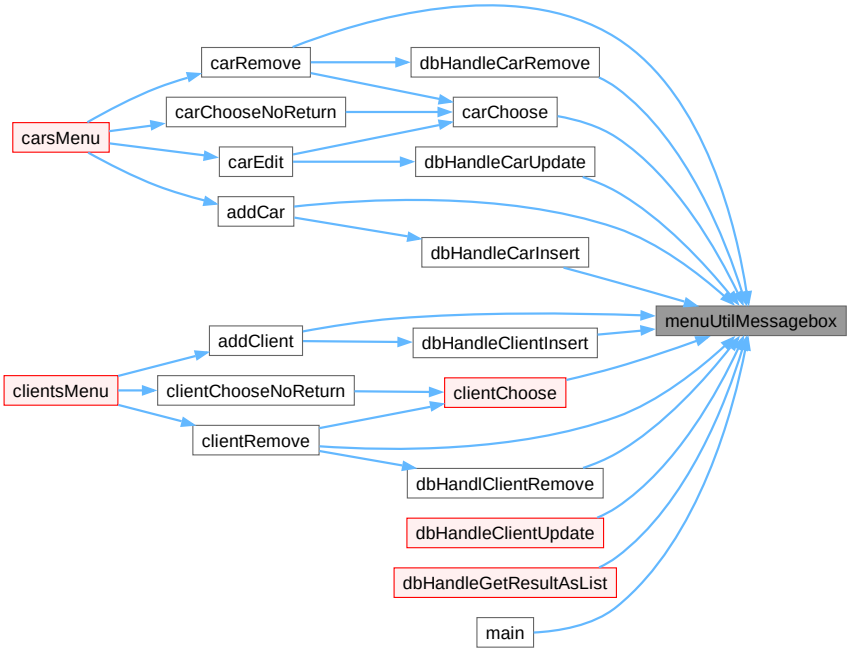
title	Title of message box
message	Array of strings each representing line of text in the message, NULL terminated array

Definition at line 99 of file menuutil.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.27.4.16 printColumnNames()

```
void printColumnNames (
            WINDOW * win,
            const char *const columnNames[],
            const int colCount,
            const int current )
```

Prints header row of table.

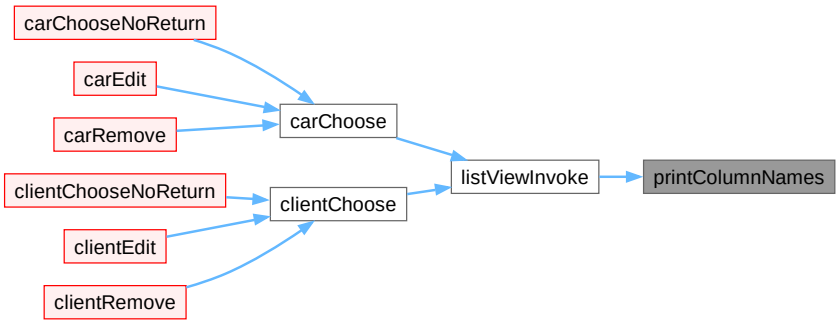
Parameters

win	Window on which it will be printed.
columnNames	Column Names passed from listViewInvoke.
colCount	colCount from listViewInvoke
current	Which column should be highlighed.

One header will be colored, indicated by current.

Definition at line 545 of file menuutil.c.

Here is the caller graph for this function:



5.27.4.17 printWindowBoarders()

```
void printWindowBoarders (
             WINDOW * window,
             const char *const title )
```

Print boarders of window.

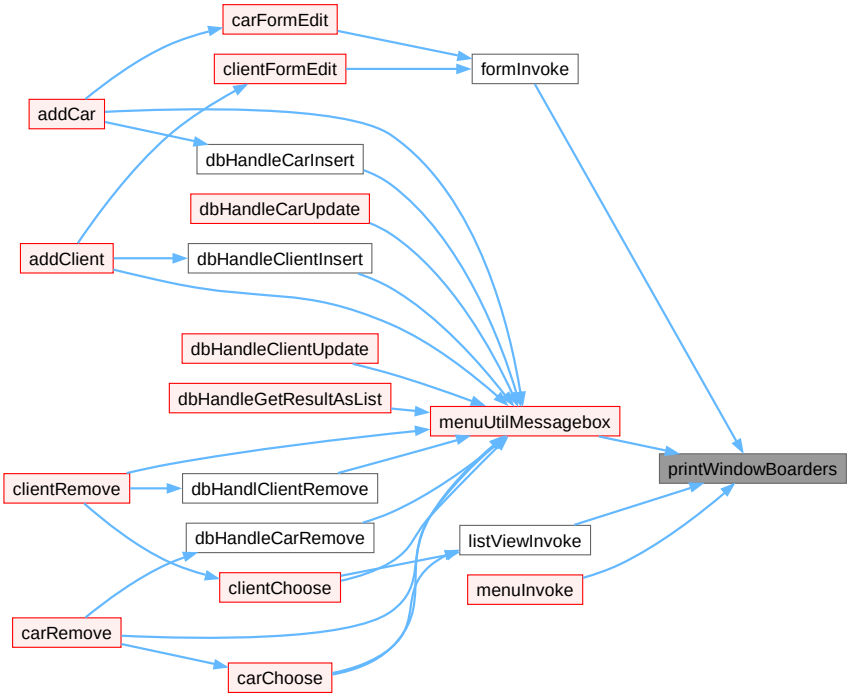
Parameters

window	WINDOW pointer
title	Char pointer to title of menu

Prints box around window, at top puts title that is also boxed, and rest of space is left unchanged.

Definition at line 85 of file menuutil.c.

Here is the caller graph for this function:



5.28 menuutil.c

Go to the documentation of this file.

```
00001 #include "menuutil.h"
00002 #include "list.h"
00003 #include <assert.h>
00004 #include <eti.h>
00005 #include <form.h>
00006 #include <math.h>
00007 #include <menu.h>
00008 #include <ncurses.h>
00009 #include <panel.h>
00010 #include <stdbool.h>
00011 #include <stdio.h>
00012 #include <stdlib.h>
00013 #include <string.h>
00014
00024 #define NOTRACE
00025
00029 #define MENUMARK (" * ")
00030
00037 int max(const int a, const int b) { return a > b ? a : b; }
00038
00045 int getLongestStringLength(const char *const stringArr[],
00046                            const int stringsCount) {
00047   assert(stringArr);
00048   assert(stringsCount > 0);
00049   int result = strlen(stringArr[0]);
00050   for (int i = 0; i < stringsCount; ++i) {
00051     if (result < strlen(stringArr[i]))
00052       result = strlen(stringArr[i]);
00053   }
00054   return result;
00055 }
00056
00064 int computeWidth(const char *const title, const char *const choices[],
00065                  const int optionsCount) {
00066   assert(choices != NULL);
00067   assert(choices[0] != NULL);
00068   // assume title is longest (don't count mark)
```

```
00069
00070   const int titleLength = strlen(title);
00071   const int choicesColsNeeded = getLongestStringLength(choices, optionsCount);
00072
00073   const int colsNeeded =
00074       titleLength > choicesColsNeeded ? titleLength : choicesColsNeeded;
00075
00076   // +2 for boarders, +2 because menu leaves 2 empty columns
00077   int windowCols = strlen(MENUMARK) + colsNeeded + 2 + 2;
00078   // make title centered
00079   if ((windowCols ^ strlen(title) & 1))
00080     ++windowCols;
00081
00082   return windowCols;
00083 }
00084
00085 void printWindowBoarders(WINDOW *window, const char *const title) {
00086   box(window, 0, 0);
00087
00088   mvwprintw(window, 1, (getmaxx(window) - (strlen(title))) / 2, "%s", title);
00089   mvwaddch(window, 2, 0, ACS_LTEE);
00090   mvwhline(window, 2, 1, ACS_HLINE, getmaxx(window) - 2);
00091   mvwaddch(window, 2, getmaxx(window) - 1, ACS_RTEE);
00092 }
00093
00099 void menuUtilMessagebox(const char *const title, const char *const message[]) {
00100   int rowsNeeded = 0;
00101   if (message != NULL) {
00102     for (int i = 0; message[i] != NULL; ++i) {
00103       ++rowsNeeded;
00104     }
00105   }
00106   int messWinWidth =
00107       2 +
00108       max((message ? getLongestStringLength(message, rowsNeeded) : rowsNeeded),
00109           strlen(title));
00110   int messWinHeight = 4 + rowsNeeded;
00111   WINDOW *messWin =
00112       newwin(messWinHeight, messWinWidth, (LINES - messWinHeight) / 2,
00113              (COLS - messWinWidth) / 2);
00114   PANEL *panel = new_panel(messWin);
00115   printWindowBoarders(messWin, title);
00116   for (int i = 0; i < rowsNeeded; ++i) {
00117     mvwprintw(messWin, 3 + i, 1, "%s", message[i]);
00118   }
00119   update_panels();
00120   doupdate();
00121   getch();
00122   del_panel(panel);
00123   delwin(messWin);
00124   update_panels();
00125   doupdate();
00126 }
00132 static void menuHandleIteraction(MENU *menu, PANEL *panel) {
00133   bool doExit = FALSE;
00134   do {
00135     update_panels();
00136     doupdate();
00137     int input = getch();
00138     switch (input) {
00139     case KEY_UP:
00140       menu_driver(menu, REQ_UP_ITEM);
00141       break;
00142     case KEY_DOWN:
00143       menu_driver(menu, REQ_DOWN_ITEM);
00144       break;
00145     case 10:;
00146       ITEM *curitem = current_item(menu);
00147 #ifndef NOTRACE
00148       const char *const name = item_name(curitem);
00149       // printing choices on stdscr for testing.
00150       move(LINES - 2, 0);
00151       clrtoeol();
00152       attron(COLOR_PAIR(1));
00153       printw("SELECTED: %s", name);
00154       attroff(COLOR_PAIR(1));
00155 #endif
00156       // EXIT has null pointer, break the switch and loop
00157       if (item_userptr(curitem) == NULL) {
00158         doExit = TRUE;
00159         break;
00160       }
00161       hide_panel(panel);
00162       // cast to function
00163       ((void (*)(void))(item_userptr(curitem)))();
00164       show_panel(panel);
00165     }
```

```
00166   } while (!doExit);
00167 }
00168
00179 void menuInvoke(const char *const title, const char *const choices[],
00180                 const int choicesCount, void (*menuFun[])(void)) {
00181   // Instantiate items for menu
00182   ITEM **mainMenuItems = calloc(choicesCount + 1, sizeof(ITEM *));
00183   for (int i = 0; i < choicesCount; ++i) {
00184     mainMenuItems[i] = new_item(choices[i], choices[i]);
00185     set_item_userptr(mainMenuItems[i], menuFun[i]);
00186   }
00187   mainMenuItems[choicesCount] = NULL;
00188
00189   const int windowCols = computeWidth(title, choices, choicesCount);
00190   // boarders(3) + title(1) + choices count
00191   const int windowRows = choicesCount + 4;
00192   WINDOW *mainMenuWindow =
00193       newwin(windowRows, windowCols, (LINES - windowRows) / 2,
00194              (COLS - windowCols) / 2);
00195   keypad(mainMenuWindow, TRUE);
00196   PANEL *panel = new_panel(mainMenuWindow);
00197   MENU *mainMenu = new_menu(mainMenuItems);
00198   set_menu_win(mainMenu, mainMenuWindow);
00199   // -4 for boarders and title, start leave 3 lines for
00200   // boarders and title , and leave left boarder alone.
00201   set_menu_sub(mainMenu,
00202                derwin(mainMenuWindow, choicesCount, windowCols - 4, 3, 1));
00203   set_menu_mark(mainMenu, MENUMARK);
00204   set_menu_items(mainMenu, mainMenuItems);
00205   menu_opts_off(mainMenu, O_SHOWDESC);
00206
00207   printWindowBoarders(mainMenuWindow, title);
00208
00209   post_menu(mainMenu);
00210
00211   menuHandleIteraction(mainMenu, panel);
00212
00213   // Deallocation
00214   unpost_menu(mainMenu);
00215   del_panel(panel);
00216   delwin(menu_sub(mainMenu));
00217   delwin(menu_win(mainMenu));
00218   free_menu(mainMenu);
00219   for (int i = 0; i < choicesCount; ++i)
00220     free_item(mainMenuItems[i]);
00221   free(mainMenuItems);
00222 }
00223
00229 static void formHandleIteraction(FORM *form) {
00230   bool doExit = false;
00231   int tmp; // < C98 moment
00232
00233   do {
00234     update_panels();
00235     doupdate();
00236     int input = getch();
00237     switch (input) {
00238     case 10:
00239       tmp = form_driver(form, REQ_DOWN_FIELD);
00240 #ifndef NOTRACE
00241       attron(COLOR_PAIR(1));
00242       mvprintw(LINES - 4, 0, "form_driver status code = %d", tmp);
00243       attroff(COLOR_PAIR(1));
00244 #endif
00245       if (tmp == E_INVALID_FIELD) {
00246         break;
00248       }
00249       doExit = true;
00250       form_driver(form, REQ_END_LINE);
00251       break;
00252     case KEY_DOWN:
00253       form_driver(form, REQ_DOWN_FIELD);
00254       form_driver(form, REQ_END_LINE);
00255       break;
00256     case KEY_UP:
00257       form_driver(form, REQ_UP_FIELD);
00258       form_driver(form, REQ_END_LINE);
00259       break;
00260     case KEY_LEFT:
00261       form_driver(form, REQ_PREV_CHAR);
00262       break;
00263     case KEY_RIGHT:
00264       form_driver(form, REQ_NEXT_CHAR);
00265       break;
00266     // Delete the char before cursor
00267     case KEY_BACKSPACE:
00268     case 127:
```

```
00269       form_driver(form, REQ_DEL_PREV);
00270       break;
00271     // Delete the char under the cursor
00272     case KEY_DC:
00273       form_driver(form, REQ_DEL_CHAR);
00274       break;
00275     default:
00276       form_driver(form, input);
00277       break;
00278     }
00279   } while (!doExit);
00280 }
00281
00289 void formInvoke(FORM *form, const char *const formFieldNames[],
00290                 const char *const title) {
00291   assert(form);
00292   assert(title);
00293   assert(formFieldNames);
00294   // form sub window rows
00295   int subRows;
00296   // form sub window cols
00297   int subCols;
00298   scale_form(form, &subRows, &subCols);
00299
00300   const int titleLenght = strlen(title);
00301   // How many columns are needed for field names.
00302   const int fieldNamesColsNeeded =
00303       getLongestStringLength(formFieldNames, field_count(form));
00304   // +4 for boarders
00305   const int formWinCols =
00306       max(titleLenght, fieldNamesColsNeeded + FORMFIELDLENGTH) + 4;
00307   // Rows will be rows needed for fields + 3 for boarders + 1 row for title
00308   const int formWinRows = subRows + 3 + 1;
00309
00310   WINDOW *formWin = newwin(formWinRows, formWinCols, (LINES - formWinRows) / 2,
00311                            (COLS - formWinCols) / 2);
00312   keypad(formWin, true);
00313   set_form_win(form, formWin);
00314   // start at 3 (2 boarders + title row)
00315   // +1 space for nice looking
00316   set_form_sub(form, derwin(form_win(form), subRows, subCols, 3,
00317                             fieldNamesColsNeeded + 1));
00318
00319   PANEL *panel = new_panel(formWin);
00320
00321   printWindowBoarders(form_win(form), title);
00322   for (int i = 0; i < field_count(form); ++i) {
00323     mvwprintw(form_win(form), 3 + i * 2, 1, "%s", formFieldNames[i]);
00324   }
00325   post_form(form);
00326
00327   formHandleIteraction(form);
00328
00329   del_panel(panel);
00330   delwin(form_sub(form));
00331   delwin(form_win(form));
00332   unpost_form(form);
00333 #ifndef NOTRACE
00334   // print content of form on screen.
00335   attron(COLOR_PAIR(1));
00336   mvprintw(1, 1, "%s", title);
00337   for (int i = 0; i < field_count(form); ++i) {
00338     mvprintw(2 + i, 1, "%s (changed: %s): %s", formFieldNames[i],
00339              field_status(form_fields(form)[i]) ? "yes" : "no",
00340              field_buffer(form_fields(form)[i], 0));
00341   }
00342   attroff(COLOR_PAIR(1));
00343 #endif
00344 }
00345
00351 FORM *formInit(const int fieldCount) {
00352   // allocate
00353   FIELD **field = calloc(sizeof(FIELD *), fieldCount + 1);
00354   field[fieldCount] = NULL;
00355   for (int i = 0; i < fieldCount; ++i) {
00356     field[i] = new_field(1, FORMFIELDLENGTH, 2 * i, 1, 0, 0);
00357     assert(field[i]);
00358     set_field_back(field[i], A_UNDERLINE);
00359     field_opts_off(field[i], O_AUTOSKIP);
00360   }
00361   FORM *form = new_form(field);
00362
00363   return form;
00364 }
00365
00370 void formFree(FORM *form) {
00371   unpost_form(form);
```

```
00372   FIELD **fields = form_fields(form);
00373   const int fieldCount = field_count(form);
00374   free_form(form);
00375   for (int i = 0; i < fieldCount; ++i) {
00376     free_field(fields[i]);
00377   }
00378   free(fields);
00379 }
00380
00383 enum ListViewIteractionStateCode {
00385   sortNext,
00387   sortPrev,
00389   sortInvert,
00391   canceled,
00393   chosen,
00395   invalid,
00396 };
00397
00403 static enum ListViewIteractionStateCode
00404 listViewHandleIteraction(struct ListNode **result, MENU *menu) {
00405   bool doExit = FALSE;
00406   enum ListViewIteractionStateCode state = invalid;
00407
00408   do {
00409     update_panels();
00410     doupdate();
00411     int input = getch();
00412
00413     switch (input) {
00414     case KEY_UP:
00415       menu_driver(menu, REQ_UP_ITEM);
00416       break;
00417     case KEY_DOWN:
00418       menu_driver(menu, REQ_DOWN_ITEM);
00419       break;
00420     case 10:
00421       *result = item_userptr(current_item(menu));
00422       state = chosen;
00423       doExit = true;
00424       break;
00425     case KEY_F(1):
00426     case 'q':
00427       state = canceled;
00428       doExit = true;
00429       break;
00430     case KEY_RIGHT:
00431       state = sortNext;
00432       doExit = true;
00433       break;
00434     case KEY_LEFT:
00435       state = sortPrev;
00436       doExit = true;
00437       break;
00438     case 's':
00439       state = sortInvert;
00440       doExit = true;
00441       break;
00442     case 'r':
00443       state = sortInvert;
00444       doExit = true;
00445       break;
00446     default:
00447       break;
00448     };
00449   } while (!doExit);
00450   return state;
00451 }
00452
00459 static void listViewFreeList(struct List **list, void (*dealloactor)(void *)) {
00460   while (listSize(*list) > 0) {
00461     dealloactor(listGetFront(*list)->m_data);
00462     listDeleteNode(*list, listGetFront(*list));
00463   }
00464   assert(listGetFront(*list) == 0);
00465   assert(listGetBack(*list) == NULL);
00466   free(*list);
00467   list = NULL;
00468 }
00469
00485 static MENU *listViewInitMenu(struct List *list, char *(*getItemString)(void *),
00486                               const int colCount) {
00487   ITEM **items = calloc(listSize(list) + 1, sizeof(ITEM *));
00488   struct ListNode *it = listGetFront(list);
00489   for (int i = 0; i < listSize(list); ++i) {
00490     char *itemAsStr = getItemString(it->m_data);
00491     items[i] = new_item(itemAsStr, itemAsStr);
00492     set_item_userptr(items[i], it);
```

```
00493     it = it->m_next;
00494   }
00495   items[listSize(list)] = NULL;
00496   // boreader + size of menumark
00497   const int utilityCols = 2 + strlen(MENUMARK);
00498   // boreader + titles
00499   const int utilityLines = 2 + 1 + 1;
00500   const int listViewWindowWidth = FORMFIELDLENGTH * colCount + utilityCols;
00501   const int listViewWidnowHight = 16 + 5;
00502   WINDOW *menuWin = newwin(listViewWidnowHight, listViewWindowWidth,
00503                            (LINES - listViewWidnowHight) / 2,
00504                            (COLS - listViewWindowWidth) / 2);
00505   keypad(menuWin, true);
00506   MENU *menu = new_menu(items);
00507   set_menu_userptr(menu, items);
00508   set_menu_mark(menu, MENUMARK);
00509   set_menu_win(menu, menuWin);
00510   set_menu_sub(menu,
00511                derwin(menu_win(menu), getmaxy(menu_win(menu)) - utilityCols,
00512                       getmaxx(menuWin) - 2, utilityLines, 1));
00513   set_menu_items(menu, items);
00514   menu_opts_off(menu, O_SHOWDESC);
00515   return menu;
00516 }
00517
00524 static void listViewFreeMenu(MENU *menu, const int itemCount) {
00525   ITEM **items = menu_userptr(menu);
00526   delwin(menu_sub(menu));
00527   delwin(menu_win(menu));
00528   free_menu(menu);
00529   for (int i = 0; i < itemCount; ++i) {
00530     free((void *)item_name(items[i]));
00531     free_item(items[i]);
00532   }
00533   free(items);
00534 }
00535
00545 void printColumnNames(WINDOW *win, const char *const columnNames[],
00546                       const int colCount, const int current) {
00547   init_pair(2, COLOR_BLACK,
00548             COLOR_MAGENTA); // smh it doesn't work when it's in other file
00549   for (int i = 0; i < colCount; ++i) {
00550     if (i == current)
00551       wattron(win, COLOR_PAIR(2));
00552     mvwprintw(win, 3, strlen(MENUMARK) + 1 + i * FORMFIELDLENGTH, "%*s",
00553               FORMFIELDLENGTH, columnNames[i]);
00554     wattroff(win, COLOR_PAIR(2));
00555   }
00556 }
00557
00583 bool listViewInvoke(void **out,
00584                     void (*dataExtractor)(void **out,
00585                                           const struct ListNode *const data),
00586                     struct List *(*listFun)(int sortType, bool descending),
00587                     const char *const columnNames[], const int colCount,
00588                     char *(*getItemString)(void *),
00589                     void (*dealloactor)(void *)) {
00590   assert(listFun && "No list functions passed");
00591   assert(getItemString && "Can't create list without that function.");
00592   if (out) {
00593     assert(dataExtractor && "Can't extract data, out location provieded, but "
00594                             "extract function was not.");
00595   }
00596   if (dataExtractor)
00597     assert(out && "extractData has to have space to save data.");
00598
00599   // Load List
00600   int currentSortType = 0;
00601   bool sortDescending = false;
00602   enum ListViewIteractionStateCode choiceState = invalid;
00603   struct List *list = NULL;
00604   do {
00605     // this could be refactored at cost of readabiiltiy, but could save time
00606     // in reverse.
00607     //
00608     list = listFun(currentSortType, sortDescending);
00609
00610     assert(list);
00611     // Make list on screen.
00612     MENU *menu = listViewInitMenu(list, getItemString, colCount);
00613     PANEL *panel = new_panel(menu_win(menu));
00614     printWindowBoarders(menu_win(menu), "List viewer");
00615     printColumnNames(menu_win(menu), columnNames, colCount, currentSortType);
00616     post_menu(menu);
00617     struct ListNode *choice = NULL;
00618
00619     choiceState = listViewHandleIteraction(&choice, menu);
```

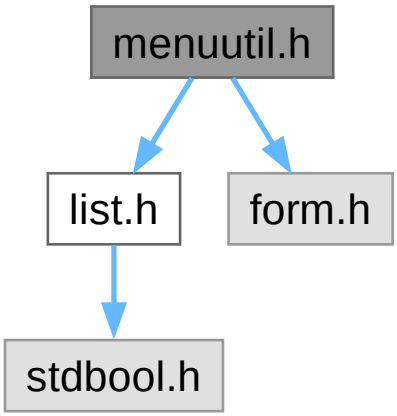
```
00620
00621     switch (choiceState) {
00622     case chosen:
00623       // if chosen choice is set already.
00624       if (dataExtractor) {
00627         dataExtractor(out, choice);
00628       }
00629       break;
00630     case canceled:
00631       break;
00632     case sortInvert:
00633       sortDescending = !sortDescending;
00634       break;
00635     case sortNext:
00636       // if current sorting is not last sorting type.
00637       if (currentSortType < colCount - 1) {
00638         ++currentSortType;
00639       }
00640       break;
00641     case sortPrev:
00642       // if current sorting is not first sorting type.
00643       if (currentSortType > 0) {
00644         --currentSortType;
00645       }
00646       break;
00647
00648     case invalid:
00649       assert(false && "Should never happen.");
00650     }
00651
00652     // free menu and list
00653     unpost_menu(menu);
00654     del_panel(panel);
00655     listViewFreeMenu(menu, listSize(list));
00656     listViewFreeList(&list, dealloactor);
00657   } while (choiceState != chosen && choiceState != canceled);
00658   return choiceState == chosen;
00659 }
```

5.29 menuutil.h File Reference

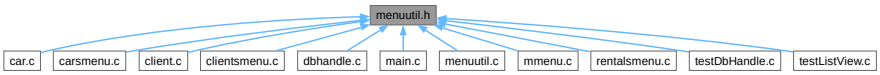
Header file for menu operations.

```
#include ¨list.h¨
#include <form.h>
```

Include dependency graph for menuutil.h:



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



Macros

- #define FORMFIELDLENGTH 40

  How many columns should forms have.

Functions

- void menuInvoke (const char ∗const title, const char ∗const choices[ ], const int choicesCount, void(∗menu↩ Fun[ ])(void))

  amount of columns of the form field.
- int getLongestStringLength (const char ∗const stringArr[ ], const int stringsCount)

  Get length of longest string in array of strings.
- void formInvoke (FORM ∗form, const char ∗const formFieldNames[ ], const char ∗const title)

  Put form on screen in nice looking form.
- FORM ∗ formInit (const int fieldCount)

  Initialize FORM.
- void formFree (FORM ∗form)

  Frees memory used for form, and it's fields.
- bool listViewInvoke (void ∗∗out, void(∗extractData)(void ∗∗out, const struct ListNode ∗const data), struct List ∗(∗listFun)(int sortType, bool descending), const char ∗const columnNames[ ], const int colCount, char ∗(∗getItemString)(void ∗), void(∗dealloactor)(void ∗))

  List Viewer for lists.
- void menuUtilMessagebox (const char ∗const title, const char ∗const message[ ])

  Displays message box on the screen with title and message.

5.29.1 Detailed Description

Header file for menu operations.

Definition in file menuutil.h.

5.29.2 Macro Definition Documentation

5.29.2.1 FORMFIELDLENGTH

```
#define FORMFIELDLENGTH 40
```

How many columns should forms have.

Also affects ListViwer.

Definition at line 17 of file menuutil.h.

5.29.3 Function Documentation

5.29.3.1 formFree()

```
void formFree (
            FORM * form )
```

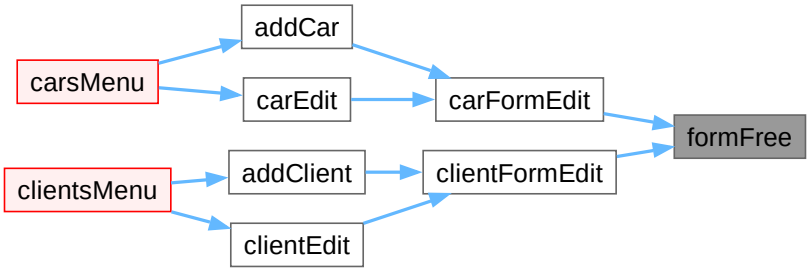
Frees memory used for form, and it's fields.

Parameters

form	Form to be free.

Definition at line 370 of file menuutil.c.

Here is the caller graph for this function:



5.29.3.2 formInit()

```
FORM * formInit (
            const int fieldCount )
```

Initialize FORM.

Parameters

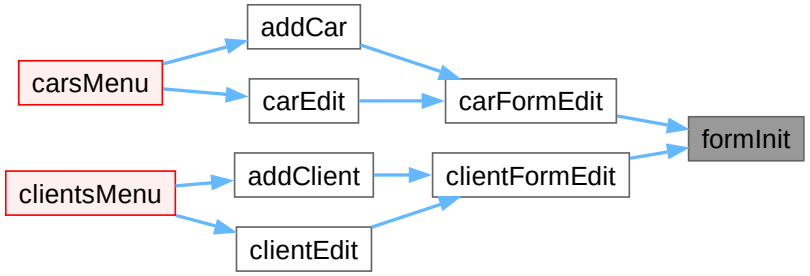
fieldCount	How many fields will be in the field.

Returns

initialized FORM pointer.

Definition at line 351 of file menuutil.c.

Here is the caller graph for this function:



5.29.3.3 formInvoke()

```
void formInvoke (
            FORM * form,
            const char *const formFieldNames[],
            const char *const title )
```

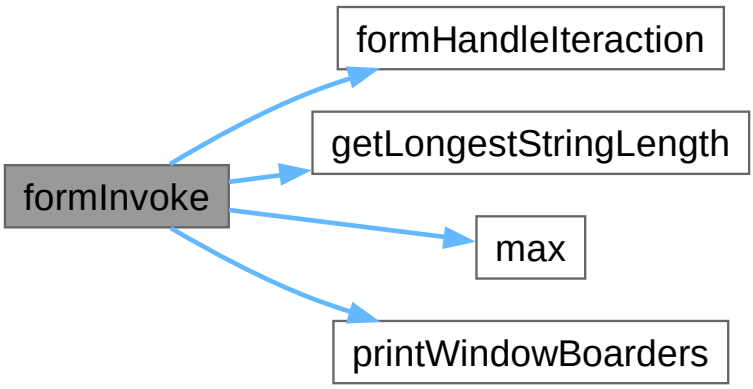
Put form on screen in nice looking form.

Parameters

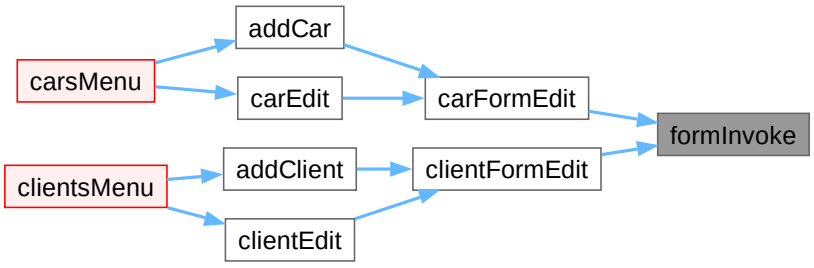
form	Form that will be put on scree.
formFieldNames	array of field names.
title	Title of window(form)

Definition at line 289 of file menuutil.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.29.3.4 getLongestStringLength()

```
int getLongestStringLength (
            const char *const stringArr[],
            const int stringsCount )
```

Get length of longest string in array of strings.

Parameters

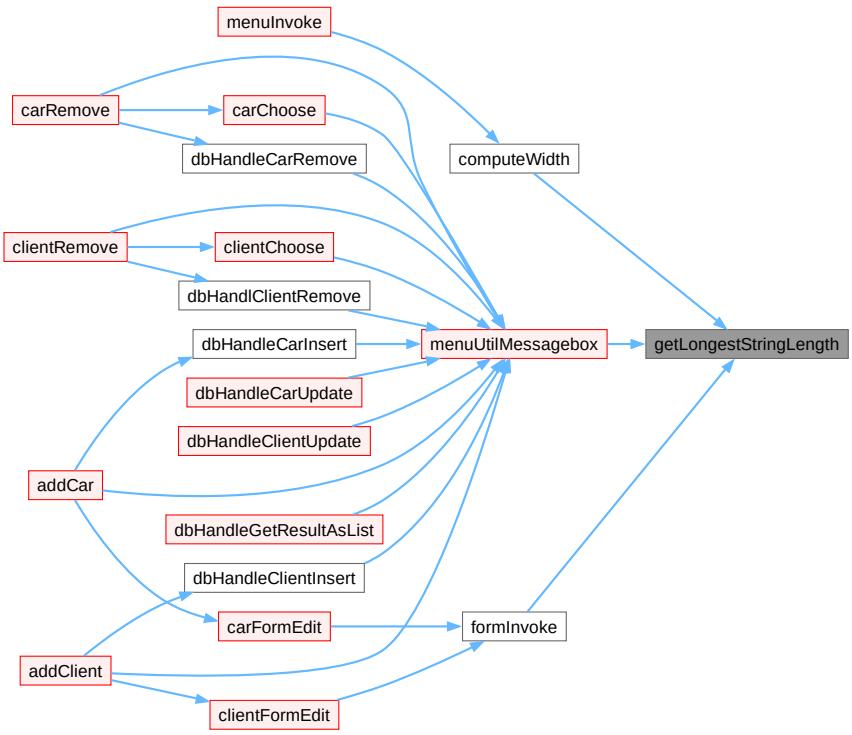
stringArr	Array of strings to look for longest string at.
stringsCount	How many strings are in the array.

Returns

Length of longest string in array.

Definition at line 45 of file menuutil.c.

Here is the caller graph for this function:



5.29.3.5 listViewInvoke()

```
bool listViewInvoke (
            void ** out,
```

```
void(*)(void **out, const struct ListNode *const data) dataExtractor,
struct List *(*)(int sortType, bool descending) listFun,
const char *const columnNames[],
const int colCount,
char *(*)(void *) getItemString,
void(*)(void *) dealloactor )
```

List Viewer for lists.

Parameters

out	Where result will be saved.
dataExtractor	Function taking two parameters first is pointer to the memory where result will be saved (out parameter will be passed internally), second is Listnode, from witch data will be extracted.

Note

dataExtractor parameter function receives pointer to internal listViewInvoke List ListNode that is deallocated after call returns so if result is to be preserved it has to do copy of data by itself.

Parameters

listFun	Functions that returns sorted list. Parameter sortType says which sort type should be used it be handled by function. Parameter descending of informs if sorting is ascending or descending.

Note

listFuns sortType paremeter should be handled in range from 0 to colCount-1.

Parameters

columnNames	array of column names strings.
colCount	How many columns are there.
getItemString	Function creating string based on ListNode::m_data(it's passed as praemeter). Should do padding.
dealloactor	Function deallocating data that is held in ListNode, not ListNode itself. Required for internal List dealocation.

Returns

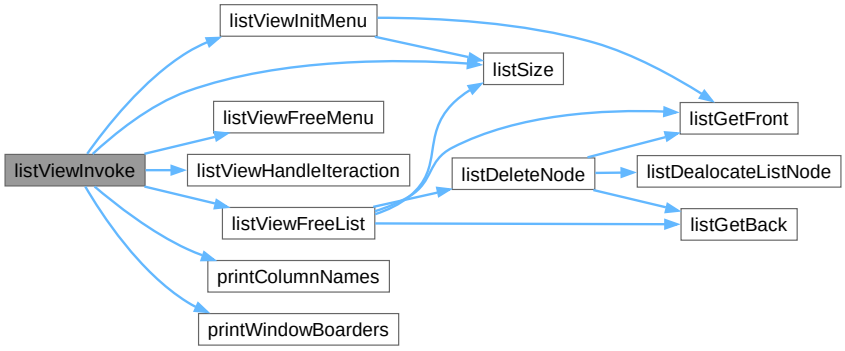
- true if chosen something.
- false if canceled.

Warning

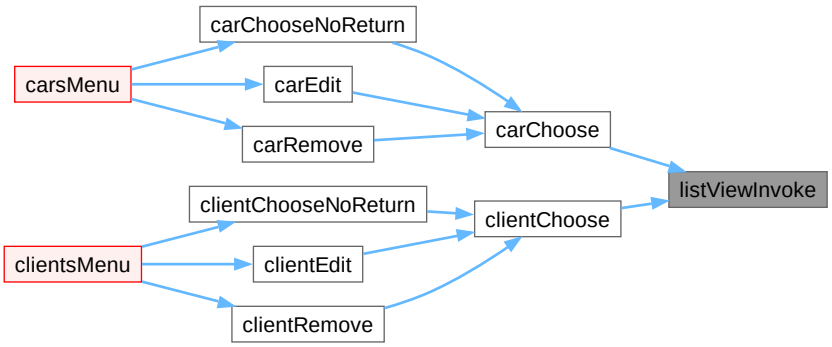
make sure extractData correctly, hard to debug.

Definition at line 583 of file menuutil.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.29.3.6 menuInvoke()

```
void menuInvoke (
            const char *const title,
            const char *const choices[],
            const int choicesCount,
            void(*[])(void) menuFun )
```

amount of columns of the form field.

amount of columns of the form field.

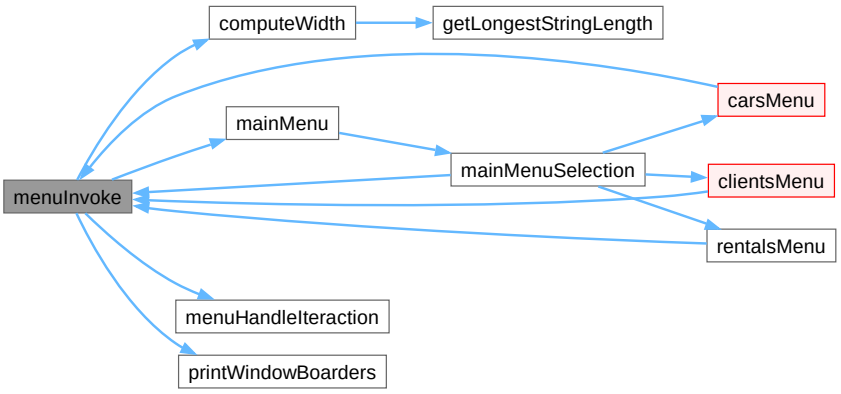
Parameters

title	Char pointer to title of menu
choices	Char pointer to table of choices
choicesCount	Number of elements in table of choices
menuFun	Table of pointers on functions

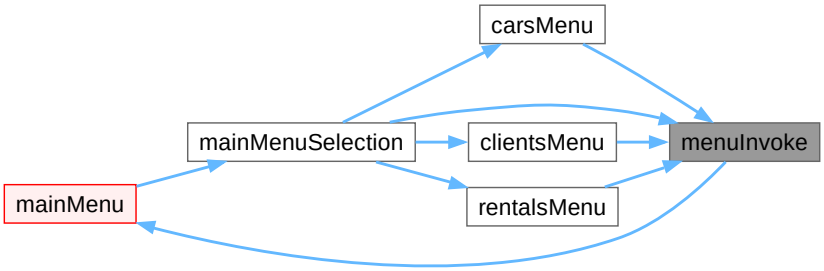
Todo Split into functions, make allocation and dallcation seperate functions, make it allocate on heap instead of stack.

Definition at line 179 of file menuutil.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.29.3.7 menuUtilMessagebox()

```
void menuUtilMessagebox (
            const char *const title,
            const char *const message[] )
```

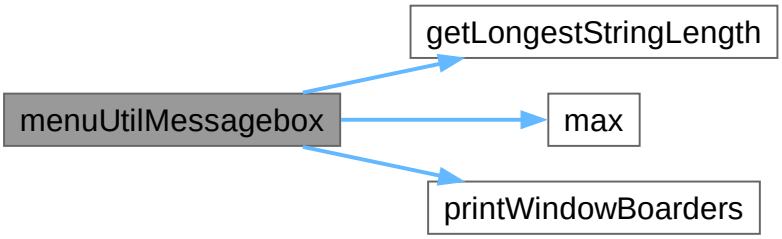
Displays message box on the screen with title and message.

Parameters

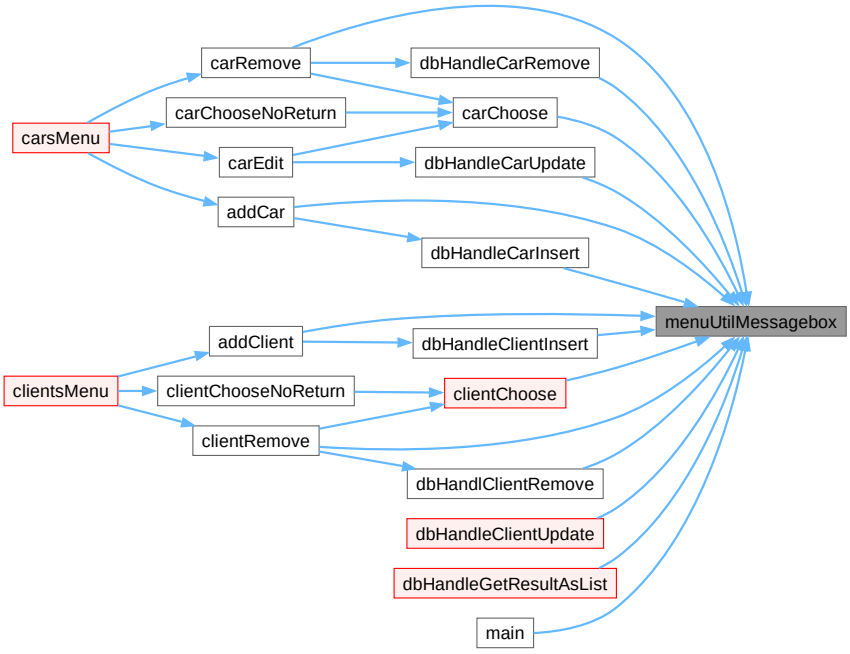
title	Title of message box
message	Array of strings each representing line of text in the message, NULL terminated array

Definition at line 99 of file menuutil.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.30 menuutil.h

Go to the documentation of this file.

```
00001 #ifndef MENUUTIL_H
00002 #define MENUUTIL_H
00003
00009 #include "list.h"
00010 #include <form.h>
00011
00017 #define FORMFIELDLENGTH 40
00018
00023 void menuInvoke(const char *const title, const char *const choices[],
00024                 const int choicesCount, void (*menuFun[])(void));
```

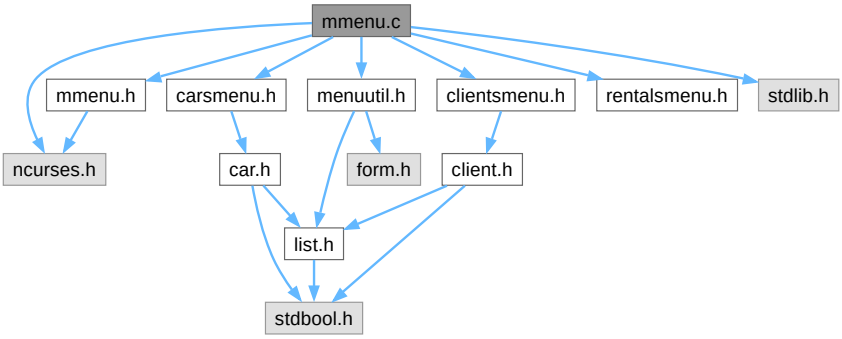
```
00025
00026 int getLongestStringLength(const char *const stringArr[],
00027                            const int stringsCount);
00028
00029 void formInvoke(FORM *form, const char *const formFieldNames[],
00030                 const char *const title);
00031
00032 FORM *formInit(const int fieldCount);
00033
00034 void formFree(FORM *form);
00035
00036 bool listViewInvoke(void **out,
00037                     void (*extractData)(void **out,
00038                                         const struct ListNode *const data),
00039                     struct List *(*listFun)(int sortType, bool descending),
00040                     const char *const columnNames[], const int colCount,
00041                     char *(*getItemString)(void *),
00042                     void (*dealloactor)(void *));
00043
00044 void menuUtilMessagebox(const char *const title, const char *const message[]);
00045
00046 #endif // MENUUTIL_H
```

5.31 mmenu.c File Reference

Menu implementation.

```
#include ¨mmenu.h¨
#include ¨carsmenu.h¨
#include ¨clientsmenu.h¨
#include ¨menuutil.h¨
#include ¨rentalsmenu.h¨
#include <ncurses.h>
#include <stdlib.h>
```

Include dependency graph for mmenu.c:



Functions

- void mainMenuSelection (void)

  Handles main menu.
- void mainMenu (void)

  Invokes menu.

5.31.1 Detailed Description

Menu implementation.

Definition in file mmenu.c.

5.31.2 Function Documentation

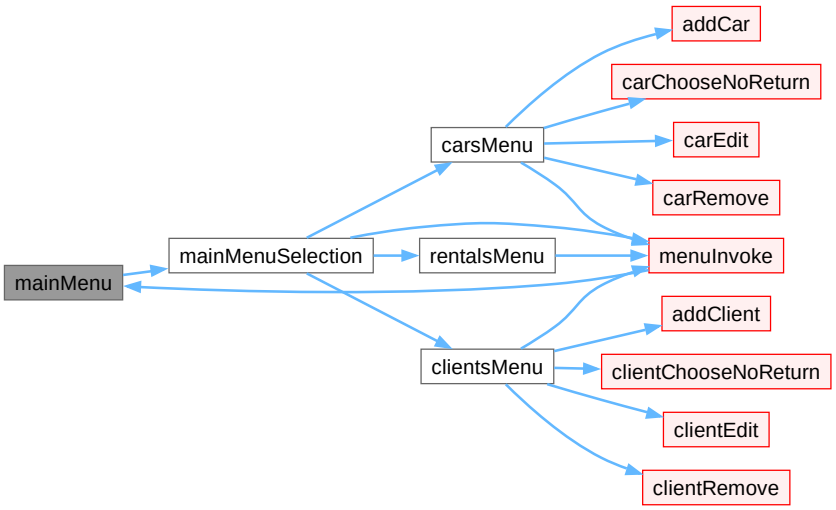
5.31.2.1 mainMenu()

```
void mainMenu (
            void  )
```

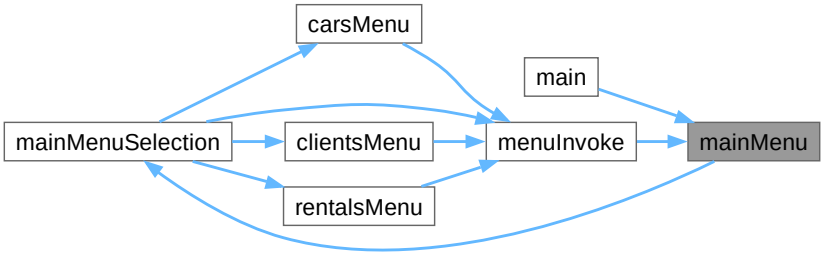
Invokes menu.

Definition at line 27 of file mmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:



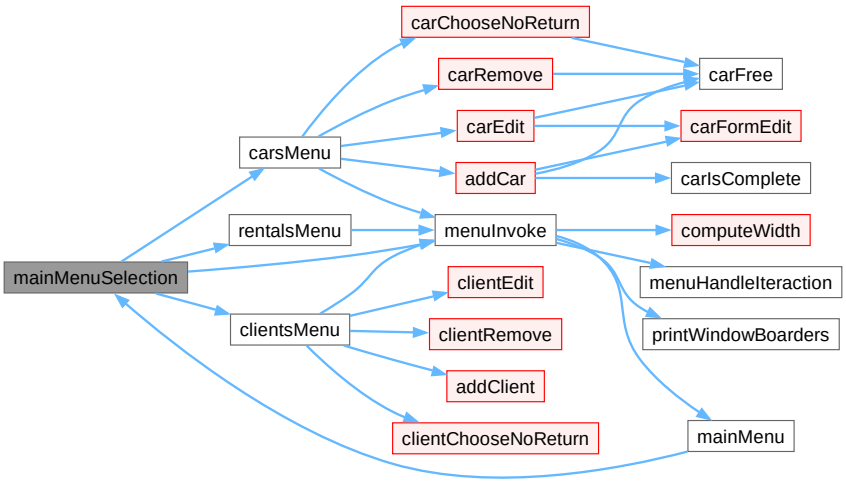
5.31.2.2 mainMenuSelection()

```
void mainMenuSelection (
            void  )
```

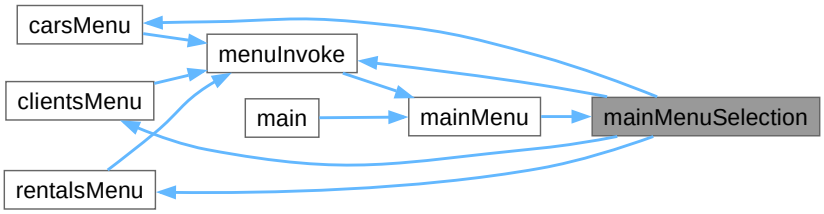
Handles main menu.

Definition at line 19 of file mmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.32 mmenu.c

Go to the documentation of this file.

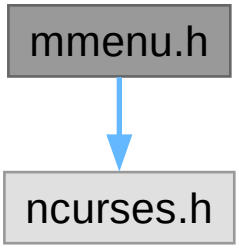
```
00001 #include "mmenu.h"
00002 #include "carsmenu.h"     //For submenu.
00003 #include "clientsmenu.h" //For submenu.
00004 #include "menuutil.h"     //For displaying
00005 #include "rentalsmenu.h" //For submenu.
00006 #include <ncurses.h>      //For displaying.
00007 #include <stdlib.h>       //For NULL.
00008
00017 void mainMenuSelection(void);
00018
```

```
00019 void mainMenuSelection(void) {
00020   const char *const title = "Main menu";
00021   const char *const choices[] = {"Cars", "Clients", "Rentals", "Exit"};
00022   const int choicesCount = sizeof(choices) / sizeof(choices[0]);
00023   void (*menuFun[])(void) = {carsMenu, clientsMenu, rentalsMenu, NULL};
00024   menuInvoke(title, choices, choicesCount, menuFun);
00025 }
00026
00027 void mainMenu(void) {
00028   initscr();
00029   noecho();
00030   cbreak();
00031   keypad(stdscr, TRUE);
00032   start_color();
00033   init_pair(1, COLOR_BLACK, COLOR_GREEN); // debugging color
00034   mainMenuSelection();
00035   endwin();
00036 }
```

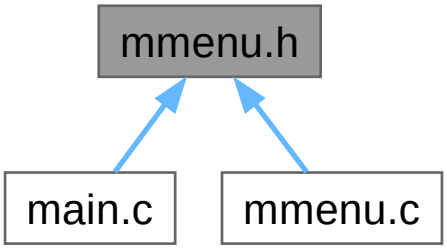
5.33 mmenu.h File Reference

For invoking menu.

`#include <ncurses.h>`
Include dependency graph for mmenu.h:



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



Functions

- void mainMenu ()

  Invokes menu.
- void printWindowBoarders (WINDOW ∗window, const char ∗const title)

  Print boarders of window.

5.33.1 Detailed Description

For invoking menu.

Definition in file mmenu.h.

5.33.2 Function Documentation

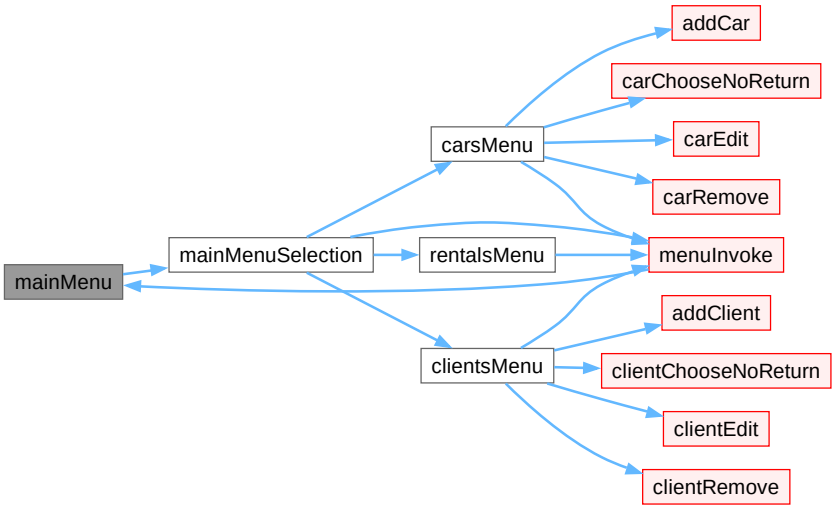
5.33.2.1 mainMenu()

```
void mainMenu ( )
```

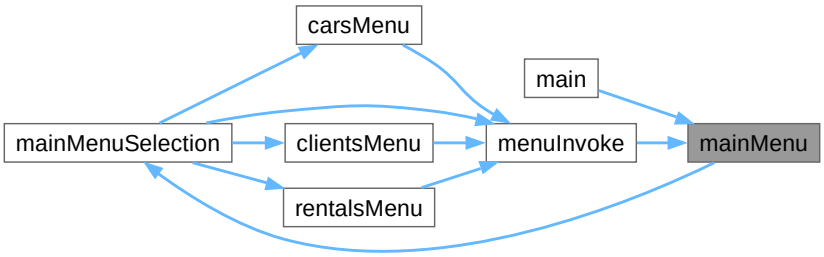
Invokes menu.

Definition at line 27 of file mmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.2.2 printWindowBoarders()

```
void printWindowBoarders (
             WINDOW * window,
             const char *const title )
```

Print boarders of window.

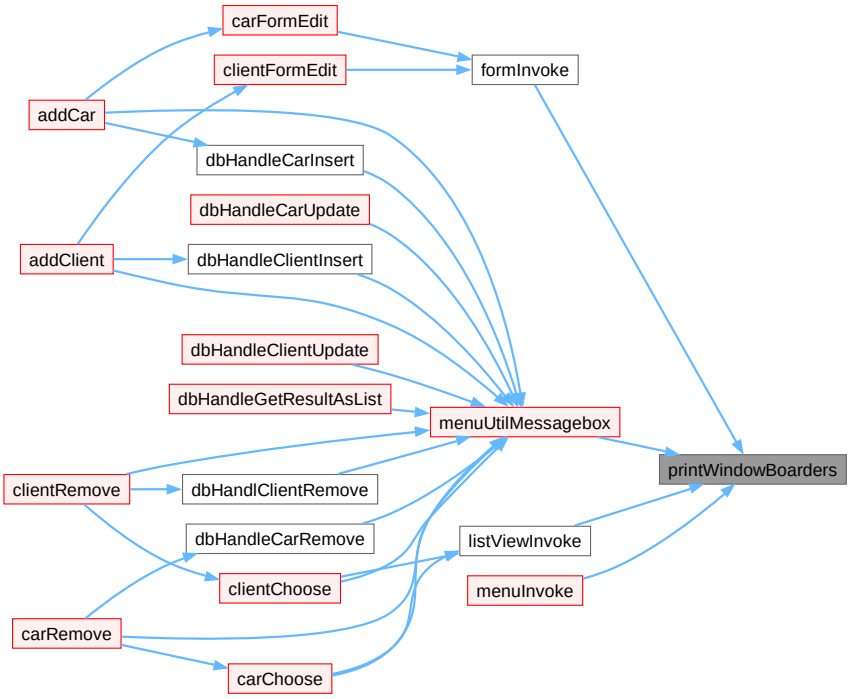
Parameters

window	WINDOW pointer
title	Char pointer to title of menu

Prints box around window, at top puts title that is also boxed, and rest of space is left unchanged.

Definition at line 85 of file menuutil.c.

Here is the caller graph for this function:



5.34 mmenu.h

Go to the documentation of this file.

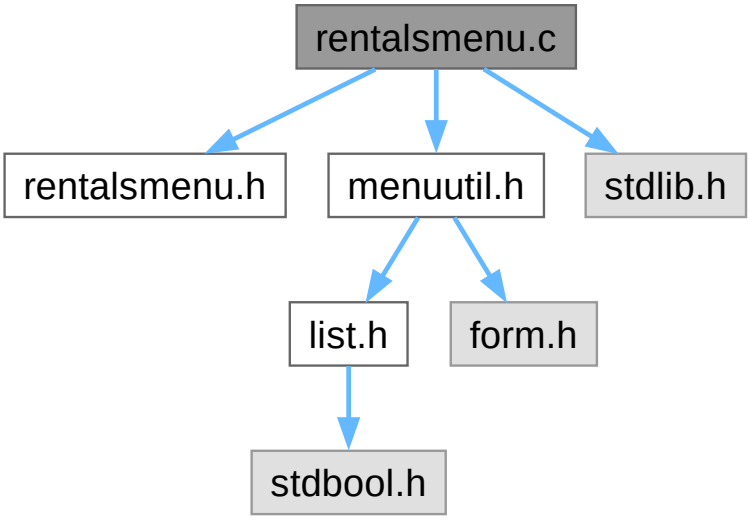
```
00001 #ifndef MMENU_H
00002 #define MMENU_H
00003 #include <ncurses.h>
00004
00013 void mainMenu();
00014
00023 void printWindowBoarders(WINDOW *window, const char *const title);
00024
00025 #endif // MMENU_H
```

5.35 rentalsmenu.c File Reference

Rentals menu implementation.

```
#include ¨rentalsmenu.h¨
#include ¨menuutil.h¨
#include <stdlib.h>
```

Include dependency graph for rentalsmenu.c:



Functions

- void rentalsMenu (void)
  Handles Rentals Menu.

5.35.1 Detailed Description

Rentals menu implementation.

Definition in file rentalsmenu.c.

5.35.2 Function Documentation

5.35.2.1 rentalsMenu()

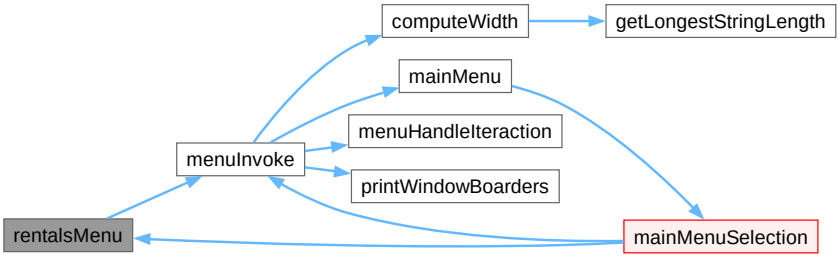
```
void rentalsMenu (
            void  )
```

Handles Rentals Menu.

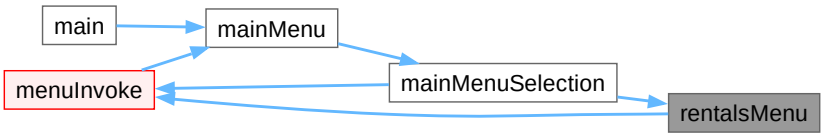
Todo implemnet submenus

Definition at line 10 of file rentalsmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.36 rentalsmenu.c

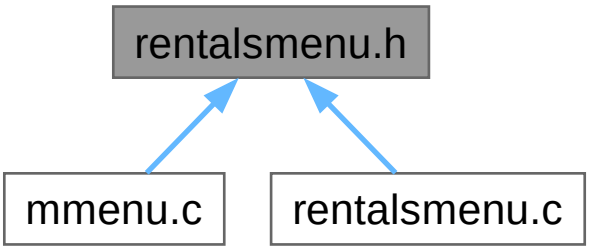
Go to the documentation of this file.

```
00001 #include "rentalsmenu.h"
00002 #include "menuutil.h"
00003 #include <stdlib.h>
00004
00010 void rentalsMenu(void) {
00011   const char *const title = "Rentals";
00012   const char *const choices[] = {"listRents", "addRent", "returnRent",
00013                                  "returnToMainMenu"};
00014   const int choicesCount = sizeof(choices) / sizeof(choices[0]);
00016   void (*menuFun[])(void) = {NULL, NULL, NULL, NULL};
00017   menuInvoke(title, choices, choicesCount, menuFun);
00018 }
```

5.37 rentalsmenu.h File Reference

Rentals menu interface.

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



Functions

- void rentalsMenu (void)
  Handles Rentals Menu.

5.37.1 Detailed Description

Rentals menu interface.

Definition in file rentalsmenu.h.

5.37.2 Function Documentation

5.37.2.1 rentalsMenu()

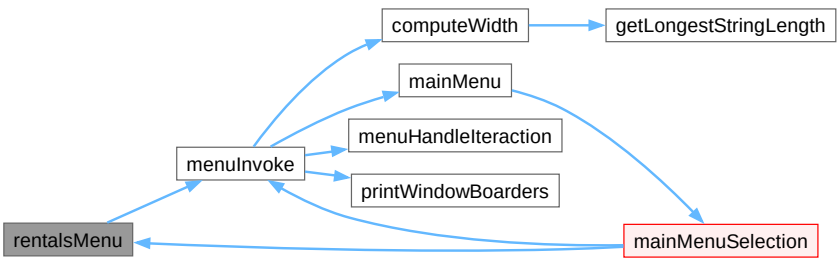
```
void rentalsMenu (
            void  )
```

Handles Rentals Menu.

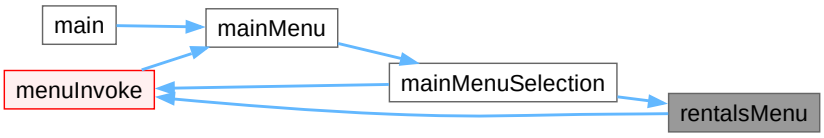
Todo implemnet submenus

Definition at line 10 of file rentalsmenu.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.38 rentalsmenu.h

Go to the documentation of this file.

```
00001 #ifndef RENTALSMENU_H
00002 #define RENTALSMENU_H
00003
00011 void rentalsMenu(void);
00012
00013 #endif // RENTALSMENU_H
```

5.39 rental.c

5.40 rental.h

```
00001 #include "client.h"
00002
00006 struct Rental {
00008   long long m_rentalID;
00010   long long m_clientID;
00012   char *m_carRegNum;
00014   char *m_since;
00016   char *m_untill;
00017 };
```

5.41 testDbHandle.c

```
00001 #include "client.h"
00002 #include "clientsmenu.h"
00003 #include "dbhandle.h"
00004 #include "list.h"
00005 #include "menuutil.h"
00006 #include "time.h"
00007 #include <ncurses.h>
00008 #include <stdio.h>
00009 #include <stdlib.h>
00010 #include <string.h>
00011
00012 int cb(void *pt, int argc, char **argv, char **colnames) {
00013   for (int i = 0; i < argc; ++i) {
00014     printf("%30s,%50s\n", colnames[i], argv[i]);
00015     printf("\n");
00016   }
00017   return 0;
00018 }
00019
00020 int cb2(void *pt, int argc, char **argv, char **colnames) {
00021   printf("%d\n", listSize(pt));
00022   struct Client *c = clientNew();
00023
00024   listPushBack(pt, c);
00025   return 0;
00026 }
00027
```

```
00028 static int cb3(void *list, int argc, char **argv, char **const colNames) {
00029   printf("list size before push is %d\n", listSize(list));
00030   struct Client *cl = clientNew();
00031   for (int i = 0; i < argc; ++i) {
00032     const char *colName = colNames[i];
00033     const char *val = argv[i];
00034     if (!strcmp(colName, "ID")) {
00035       cl->m_ID = atoi(val);
00036     } else if (!strcmp(colName, "cardID")) {
00037       cl->m_cardID = atoi(val);
00038     } else if (!strcmp(colName, "name")) {
00039       cl->m_name = calloc(FORMFIELDLENGTH + 1, sizeof(char));
00040       strcpy(cl->m_name, val);
00041     } else if (!strcmp(colName, "surname")) {
00042       cl->m_surname = calloc(FORMFIELDLENGTH + 1, sizeof(char));
00043       strcpy(cl->m_surname, val);
00044     } else if (!strcmp(colName, "phoneNumber")) {
00045       cl->m_phoneNum = atoi(val);
00046     } else if (!strcmp(colName, "adress")) {
00047       cl->m_adress = calloc(FORMFIELDLENGTH + 1, sizeof(char));
00048       strcpy(cl->m_adress, val);
00049     } else {
00050       fprintf(stderr, "Client to structure fail. FAILED on %s", colName);
00051       abort();
00052     }
00053   }
00054   listPushBack(list, cl);
00055   return 0;
00056 }
00057
00058 int main() {
00059
00060   srand(time(0));
00061   struct List *res = NULL;
00062   char *q = clientGetQueryOfSort(rand() % clientSort_MAX, 0);
00063   dbHandleGetResultAsList(&res, cb3, q);
00064   printf("Query =\t %s\n", q);
00065   printf("Result list size  = %d\n", listSize(res));
00066   struct ListNode *it = listGetFront(res);
00067
00068   while (it != NULL) {
00069     struct Client *c = it->m_data;
00070     char *str = clientGetListViewString(c);
00071     printf("%s\n", str);
00072     it = it->m_next;
00073   }
00074
00075   // not freed.
00076 }
```

5.42 testListView.c

```
00001 #include "list.h"
00002 #include "menuutil.h"
00003 #include <ncurses.h>
00004 #include <stdint.h>
00005 #include <stdio.h>
00006 #include <stdlib.h>
00007 #include <string.h>
00008
00009 bool intLess(const void *a, const void *b) {
00010   return *(int *)(a) < *(int *)(b) ? true : false;
00011 }
00012
00013 bool intMore(const void *a, const void *b) {
00014   return *(int *)(a) > *(int *)(b) ? true : false;
00015 }
00016
00017 struct List *getList(void) {
00018   // create list and insert elements.
00019   struct List *list = listCreateList();
00020   for (int i = 0; i < 100; ++i) {
00021     int *t = calloc(sizeof(int), 1);
00022     *t = i + 1;
00023     listInsert(list, t, intLess);
00024   }
00025   return list;
00026 }
00027
00028 struct List *getList2(void) {
00029   // create list and insert elements.
00030   struct List *list = listCreateList();
00031   for (int i = 0; i < 100; ++i) {
```

```
00032     int *t = calloc(sizeof(int), 1);
00033     *t = i + 1;
00034     listInsert(list, t, intMore);
00035   }
00036   return list;
00037 }
00038
00039 struct List *listGetter(int sType, bool desc) {
00040   if (sType == 0)
00041     return getList();
00042   else if (sType == 1)
00043     return getList2();
00044   else
00045     abort();
00046 }
00047
00048 char *getIntString(void *x) {
00049   char *result = calloc(sizeof(char), 2 * FORMFIELDLENGTH + 1);
00050   sprintf(result, "%*d%*d", FORMFIELDLENGTH, *(int *)x, FORMFIELDLENGTH,
00051           *(int *)x);
00052   return result;
00053 }
00054
00055 void extract(void **out, const struct ListNode *const data) {
00056   int *result = calloc(1, sizeof(int));
00057   *result = *(int *)data->m_data;
00058   *out = result;
00059 }
00060
00061 void intDel(int *data) { free(data); }
00062
00063 void randomListExample(void) {
00064   const char *const colNames[] = {"C is bad", "Lua is better"};
00065   const int colCount = sizeof(colNames) / sizeof(*colNames);
00066   int *out = NULL;
00067   // invoke choice
00068   bool didChose =
00069       listViewInvoke((void **)&out, extract, listGetter, colNames, colCount,
00070                      getIntString, (void (*)(void *))intDel);
00071
00072   if (didChose) {
00073     printf("Value of chosen element = %d", *out);
00074   } else {
00075     printf("Canceled.");
00076   }
00077
00078   free(out);
00079 }
00080
00081 int main() {
00082   initscr();
00083   noecho();
00084   cbreak();
00085   keypad(stdscr, TRUE);
00086   curs_set(0);
00087
00088   start_color();
00089   init_pair(1, COLOR_BLACK, COLOR_GREEN); // debugging color
00090   randomListExample();
00091   endwin();
00092 }
```

Index