

Borne de recharge

2.4

Généré par Doxygen 1.15.0

1 Index des structures de données	1
1.1 Structures de données	1
2 Index des fichiers	3
2.1 Liste des fichiers	3
3 Documentation des structures de données	5
3.1 Référence de la structure Client	5
3.1.1 Description détaillée	5
3.1.2 Documentation des champs	5
3.1.2.1 badge	5
3.1.2.2 nom	5
4 Documentation des fichiers	7
4.1 Référence du fichier baseclient.c	7
4.1.1 Description détaillée	8
4.1.2 Documentation des fonctions	8
4.1.2.1 base_client_authentification()	8
4.1.2.2 base_client_creer_client()	9
4.1.2.3 base_client_modifier_client()	10
4.1.2.4 base_client_supprimer_client()	10
4.1.3 Documentation des variables	10
4.1.3.1 badge	10
4.2 baseclient.c	11
4.3 Référence du fichier baseclient.h	12
4.3.1 Documentation des fonctions	13
4.3.1.1 base_client_authentification()	13
4.3.1.2 base_client_creer_client()	14
4.3.1.3 base_client_modifier_client()	15
4.3.1.4 base_client_supprimer_client()	15
4.4 baseclient.h	15
4.5 Référence du fichier borne.c	16
4.5.1 Description détaillée	16
4.5.2 Documentation des fonctions	17
4.5.2.1 main()	17
4.5.3 CLIENT RECONNU	17
4.6 borne.c	18
4.7 Référence du fichier boutons.c	23
4.7.1 Description détaillée	24
4.7.2 Documentation des fonctions	24
4.7.2.1 bouton_appuyer_pour_charger()	24
4.7.2.2 bouton_appuyer_pour_stop()	24
4.7.2.3 bouton_ensure_io()	25

4.7.2.4 bouton_relacher_pour_charger()	25
4.7.2.5 bouton_relacher_stop()	25
4.7.3 Documentation des variables	26
4.7.3.1 io_b	26
4.7.3.2 shmid_b	26
4.8 boutons.c	26
4.9 Référence du fichier boutons.h	27
4.9.1 Documentation des fonctions	27
4.9.1.1 bouton_appuyer_pour_charger()	27
4.9.1.2 bouton_appuyer_pour_stop()	28
4.9.1.3 bouton_ensure_io()	28
4.9.1.4 bouton_relacher_pour_charger()	28
4.9.1.5 bouton_relacher_stop()	29
4.10 boutons.h	29
4.11 Référence du fichier generateur_save.c	29
4.11.1 Description détaillée	30
4.11.2 Documentation des fonctions	31
4.11.2.1 generateur_save_ac()	31
4.11.2.2 generateur_save_ac_cl()	31
4.11.2.3 generateur_save_check_io()	32
4.11.2.4 generateur_save_contacteur()	32
4.11.2.5 generateur_save_contacteur_fermer()	33
4.11.2.6 generateur_save_ensure_io()	33
4.11.2.7 generateur_save_generer_dc()	34
4.11.3 Documentation des variables	34
4.11.3.1 io_g	34
4.11.3.2 shmid_g	34
4.12 generateur_save.c	35
4.13 Référence du fichier generateur_save.h	36
4.13.1 Documentation des fonctions	36
4.13.1.1 generateur_save_ac()	36
4.13.1.2 generateur_save_ac_cl()	37
4.13.1.3 generateur_save_check_io()	37
4.13.1.4 generateur_save_contacteur()	38
4.13.1.5 generateur_save_contacteur_fermer()	38
4.13.1.6 generateur_save_ensure_io()	39
4.13.1.7 generateur_save_generer_dc()	39
4.14 generateur_save.h	40
4.15 Référence du fichier global.c	40
4.15.1 Documentation des variables	41
4.15.1.1 io_b	41
4.15.1.2 shmid_b	41

4.16 global.c	41
4.17 Référence du fichier global.h	41
4.17.1 Documentation des variables	42
4.17.1.1 io_b	42
4.17.1.2 shmid_b	42
4.18 global.h	42
4.19 Référence du fichier lecteurcarte.c	42
4.19.1 Description détaillée	43
4.19.2 Documentation des fonctions	43
4.19.2.1 lecteurcarte_initialiser()	43
4.19.2.2 lecteurcarte_lire_carte()	44
4.19.2.3 lecteurcarte_retirer_carte()	44
4.19.3 Documentation des variables	45
4.19.3.1 numero	45
4.20 lecteurcarte.c	45
4.21 Référence du fichier lecteurcarte.h	46
4.21.1 Documentation des fonctions	46
4.21.1.1 lecteur_carte_demande_creation_client()	46
4.21.1.2 lecteur_carte_demande_modifier_client()	46
4.21.1.3 lecteur_carte_demande_supprimer_client()	47
4.21.1.4 lecteur_carte_valide_carte()	47
4.21.1.5 lecteurcarte_initialiser()	47
4.21.1.6 lecteurcarte_lire_carte()	47
4.21.1.7 lecteurcarte_retirer_carte()	47
4.22 lecteurcarte.h	48
4.23 Référence du fichier prise.c	48
4.23.1 Description détaillée	49
4.23.2 Documentation des fonctions	50
4.23.2.1 prise_check_io()	50
4.23.2.2 prise_deverouiller_trap()	50
4.23.2.3 prise_ensure_io()	50
4.23.2.4 prise_led_prise_off()	51
4.23.2.5 prise_led_prise_on()	52
4.23.2.6 prise_verouiller_trap()	52
4.23.3 Documentation des variables	53
4.23.3.1 io_p	53
4.23.3.2 shmid_p	53
4.24 prise.c	53
4.25 Référence du fichier prise.h	54
4.25.1 Documentation des fonctions	54
4.25.1.1 prise_check_io()	54
4.25.1.2 prise_deverouiller_trap()	55

4.25.1.3 prise_ensure_io()	55
4.25.1.4 prise_led_prise_off()	56
4.25.1.5 prise_led_prise_on()	56
4.25.1.6 prise_verouiller_trap()	57
4.25.1.7 prise_verouiller_trappe()	58
4.26 prise.h	58
4.27 Référence du fichier voyants.c	58
4.27.1 Description détaillée	59
4.27.2 Documentation des fonctions	59
4.27.2.1 voyants_ensure_io()	59
4.27.2.2 voyants_init()	60
4.27.2.3 voyants_set_charge_off()	60
4.27.2.4 voyants_set_charge_rouge()	60
4.27.2.5 voyants_set_charge_vert()	61
4.27.2.6 voyants_set_default_off()	61
4.27.2.7 voyants_set_dispo_off()	61
4.27.2.8 voyants_set_dispo_on()	62
4.27.2.9 voyants_toggle()	62
4.27.2.10 voyants_toggle_default()	62
4.27.3 Documentation des variables	62
4.27.3.1 io_l	62
4.27.3.2 shmid_l	63
4.28 voyants.c	63
4.29 Référence du fichier voyants.h	64
4.29.1 Documentation des fonctions	65
4.29.1.1 voyants_ensure_io()	65
4.29.1.2 voyants_init()	65
4.29.1.3 voyants_set_charge()	66
4.29.1.4 voyants_set_charge_off()	66
4.29.1.5 voyants_set_charge_rouge()	66
4.29.1.6 voyants_set_charge_vert()	66
4.29.1.7 voyants_set_default_off()	67
4.29.1.8 voyants_set_dispo_off()	67
4.29.1.9 voyants_set_dispo_on()	67
4.29.1.10 voyants_toggle()	68
4.29.1.11 voyants_toggle_default()	68
4.30 voyants.h	68

Chapitre 1

Index des structures de données

1.1 Structures de données

Liste des structures de données avec une brève description :

Client	5
----------------------------------	---

Chapitre 2

Index des fichiers

2.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

baseclient.c	
Programmes pour gerer le client	7
baseclient.h	12
borne.c	
Programme principal de gestion de la borne de recharge	16
boutons.c	
Programmes pour gerer les boutons sur la borne	23
boutons.h	27
generateur_save.c	
Programme pour gérer le générateur et le contacteur via la mémoire partagée	29
generateur_save.h	36
global.c	40
global.h	41
lecteurcarte.c	
Gestion du lecteur de carte utilisateur	42
lecteurcarte.h	46
prise.c	
Gestion de la prise et de la trappe de la borne	48
prise.h	54
voyants.c	
Gestion des voyants de la borne de recharge	58
voyants.h	64

Chapitre 3

Documentation des structures de données

3.1 Référence de la structure Client

Champs de données

- int [badge](#)
- char [nom](#) [50]

3.1.1 Description détaillée

[Client](#) contient deux champs; un entier pour le numero de badge et une chaine de caractere pour le nom

Définition à la ligne [23](#) du fichier [baseclient.c](#).

3.1.2 Documentation des champs

3.1.2.1 badge

```
int badge
```

Définition à la ligne [25](#) du fichier [baseclient.c](#).

3.1.2.2 nom

```
char nom[50]
```

Définition à la ligne [26](#) du fichier [baseclient.c](#).

La documentation de cette structure a été générée à partir du fichier suivant :

- [baseclient.c](#)

Chapitre 4

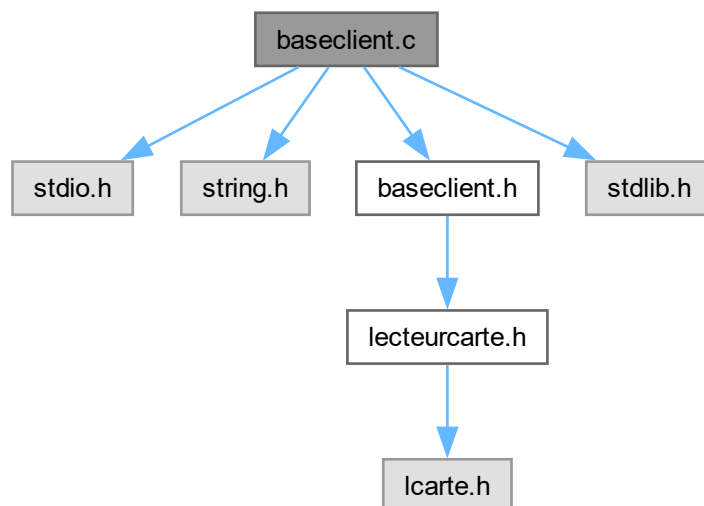
Documentation des fichiers

4.1 Référence du fichier baseclient.c

programmes pour gerer le client.

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

Graphe des dépendances par inclusion de baseclient.c:



Structures de données

— struct [Client](#)

Fonctions

— int [base_client_authentification](#) ()

fonction d'authentification du client

— void [base_client_creer_client](#) ()

fonction de creation du nouveau client en cas d'echec d'authentification

— void [base_client_modifier_client](#) ()

fonction de modification du nom d'un client existant

— void [base_client_supprimer_client](#) ()

fonction de suppression du nom et du numero de badge d'un client existant

Variables

— def badge volatile int [badge](#)

4.1.1 Description détaillée

programmes pour gerer le client.

Auteur

(Nshuti Nkurikiyinka Fidele, Mapako Eddy)

Version

1.0

Date

22 decembre 2025

programmes pour gerer le client avec une structure client et des fonctions d'authenitifiacion, creation, modification et supression

Définition dans le fichier [baseclient.c](#).

4.1.2 Documentation des fonctions

4.1.2.1 [base_client_authentification\(\)](#)

```
int base_client_authentification (  
    void )
```

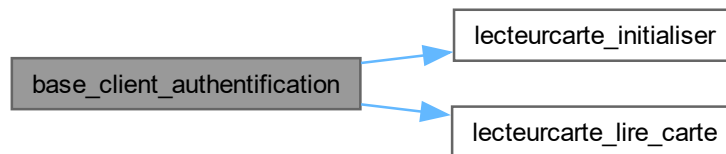
fonction d'authentification du client

Renvoie

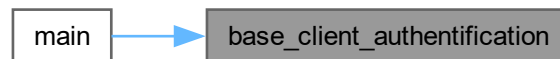
1 si le client existe et 0 en cas d'echec

Définition à la ligne 39 du fichier [baseclient.c](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

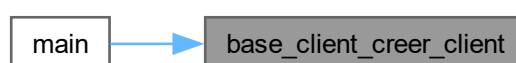
**4.1.2.2 base_client_creer_client()**

```
void base_client_creer_client (  
    void )
```

fonction de creation du nouveau client en cas d'echec d'authentification

Définition à la ligne 72 du fichier [baseclient.c](#).

Voici le graphe des appelants de cette fonction :



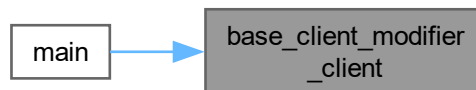
4.1.2.3 `base_client_modifier_client()`

```
void base_client_modifier_client (
    void )
```

fonction de modification du nom d'un client existant

Définition à la ligne 101 du fichier [baseclient.c](#).

Voici le graphe des appelants de cette fonction :



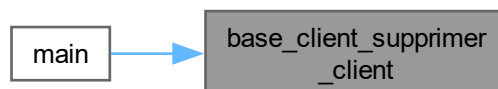
4.1.2.4 `base_client_supprimer_client()`

```
void base_client_supprimer_client (
    void )
```

fonction de suppression du nom et du numero de badge d'un client existant

Définition à la ligne 137 du fichier [baseclient.c](#).

Voici le graphe des appelants de cette fonction :



4.1.3 Documentation des variables

4.1.3.1 `badge`

```
def badge volatile int badge
```

Définition à la ligne 30 du fichier [baseclient.c](#).

4.2 baseclient.c

[Aller à la documentation de ce fichier.](#)

```

00001
00002
00013
00014 #include <stdio.h>
00015 #include <string.h>
00016 #include "baseclient.h"
00017 #include <stdlib.h>
00018
00023 typedef struct {
00024
00025     int badge;
00026     char nom[50];
00027 } Client;
00028
00029 #def badge
00030 volatile int badge;
00031
00038
00039 int base_client_authentification() {
00040
00041     Client nouveau_client;
00042     FILE *fichier_clients;
00043     lecteurcarte_initialiser();
00044     badge = lecteurcarte_lire_carte();
00045
00046     nouveau_client.badge = badge;
00047
00048     int i;
00049     fichier_clients = fopen("clients_liste.txt", "r");
00050
00051     if (fichier_clients == NULL) {
00052         printf("Erreur lors de l'ouverture du fichier.\n");
00053         return 0;
00054     }
00055     while (fscanf(fichier_clients, "%d %s", &i, nouveau_client.nom) != EOF) {
00056         if (i == nouveau_client.badge) {
00057             printf("Authentification reussie, bienvenue %s \n", nouveau_client.nom);
00058             fclose(fichier_clients);
00059             return 1;
00060         }
00061     }
00062
00063     return 0 ;
00064 }
00065
00066
00072 void base_client_creer_client() {
00073     Client nouveau_client;
00074     FILE *fichier_clients;
00075
00076     nouveau_client.badge = badge;
00077
00078     fichier_clients = fopen("clients_liste.txt", "a");
00079
00080     printf("veuillez'il vous plait renseigner votre nom svp: \n");
00081     fgets(nouveau_client.nom, sizeof(nouveau_client.nom), stdin);
00082     nouveau_client.nom[strcspn(nouveau_client.nom, "\n")] = 0; //pas de newline
00083
00084     if (fichier_clients == NULL) {
00085         printf("Erreur lors de l'ouverture du fichier.\n");
00086         return;
00087     }
00088     fprintf(fichier_clients, "%d %s\n", nouveau_client.badge, nouveau_client.nom);
00089     printf("client creer avec numero de badge %d \n", nouveau_client.badge);
00090     fclose(fichier_clients);
00091 }
00092
00093
00099
00100
00101 void base_client_modifier_client() {
00102     Client client_a_modifier;
00103     FILE *fichier_clients;
00104     FILE *fichier_temp;
00105
00106     fichier_clients = fopen("clients_liste.txt", "r");
00107     fichier_temp = fopen("temp.txt", "w");
00108
00109     if (fichier_clients == NULL || fichier_temp == NULL) {
00110         printf("Erreur lors de l'ouverture du fichier.\n");
00111         return;
00112     }

```

```

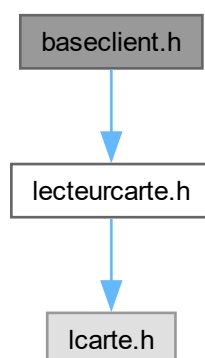
00113
00114     while (fscanf(fichier_clients, "%d %49s", &client_a_modifieur.badge, client_a_modifieur.nom) == 2) {
00115         if (client_a_modifieur.badge == badge) {
00116             printf("Entrez le nouveau nom: ");
00117             fgets(client_a_modifieur.nom, sizeof(client_a_modifieur.nom), stdin);
00118             client_a_modifieur.nom[strcspn(client_a_modifieur.nom, "\n")] = 0; //pas de newline
00119         }
00120         fprintf(fichier_temp, "%d %s\n", client_a_modifieur.badge, client_a_modifieur.nom);
00121     }
00122     fclose(fichier_clients);
00123     fclose(fichier_temp);
00124
00125     remove("clients_liste.txt");
00126     rename("temp.txt", "clients_liste.txt");
00127
00128     printf("Client modifié.\n");
00129 }
00130
00131
00132 void base_client_supprimer_client() {
00133     int badge_locale;
00134     Client client_a_supprimer;
00135     FILE *fin = fopen("clients_liste.txt", "r");
00136     FILE *fout = fopen("temp.txt", "w");
00137
00138     if (!fin || !fout) {
00139         printf("Erreur ouverture fichier.\n");
00140         return;
00141     }
00142
00143     while (fscanf(fin, "%d %49s", &badge_locale, client_a_supprimer.nom) == 2) {
00144         if (badge_locale == badge) {
00145             // on le prends pas
00146             continue;
00147         }
00148         fprintf(fout, "%d %s\n", badge_locale, client_a_supprimer.nom);
00149     }
00150
00151     fclose(fin);
00152     fclose(fout);
00153
00154     remove("clients_liste.txt");
00155     rename("temp.txt", "clients_liste.txt");
00156
00157     printf("Client supprimé.\n");
00158 }
00159
00160
00161
00162
00163
00164
00165

```

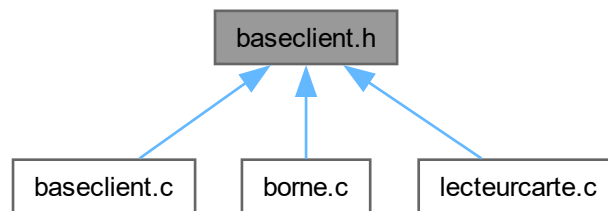
4.3 Référence du fichier baseclient.h

```
#include "lecteurcarte.h"
```

Graphe des dépendances par inclusion de baseclient.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Fonctions

— int [base_client_authentification](#) ()

fonction d'authentification du client

— void [base_client_creer_client](#) ()

fonction de creation du nouveau client en cas d'echec d'authentification

— void [base_client_modifier_client](#) ()

fonction de modification du nom d'un client existant

— void [base_client_supprimer_client](#) ()

fonction de suppression du nom et du numero de badge d'un client existant

4.3.1 Documentation des fonctions

4.3.1.1 [base_client_authentification\(\)](#)

```
int base_client_authentification (  
    void )
```

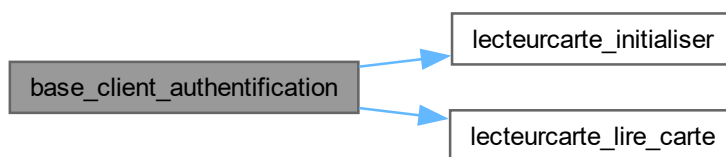
fonction d'authentification du client

Renvoie

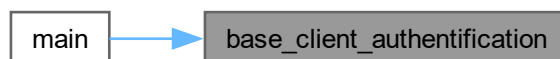
1 si le client existe et 0 en cas d'echec

Définition à la ligne 39 du fichier [baseclient.c](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



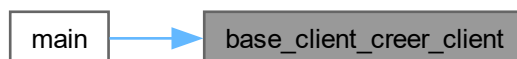
4.3.1.2 `base_client_creer_client()`

```
void base_client_creer_client ()
```

fonction de creation du nouveau client en cas d'echec d'authentification

Définition à la ligne 72 du fichier [baseclient.c](#).

Voici le graphe des appelants de cette fonction :



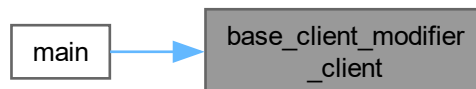
4.3.1.3 base_client_modifier_client()

```
void base_client_modifier_client ()
```

fonction de modification du nom d'un client existant

Définition à la ligne 101 du fichier [baseclient.c](#).

Voici le graphe des appelants de cette fonction :



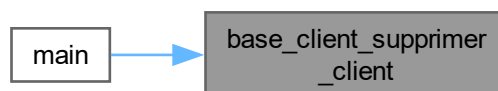
4.3.1.4 base_client_supprimer_client()

```
void base_client_supprimer_client ()
```

fonction de suppression du nom et du numero de badge d'un client existant

Définition à la ligne 137 du fichier [baseclient.c](#).

Voici le graphe des appelants de cette fonction :



4.4 baseclient.h

[Aller à la documentation de ce fichier.](#)

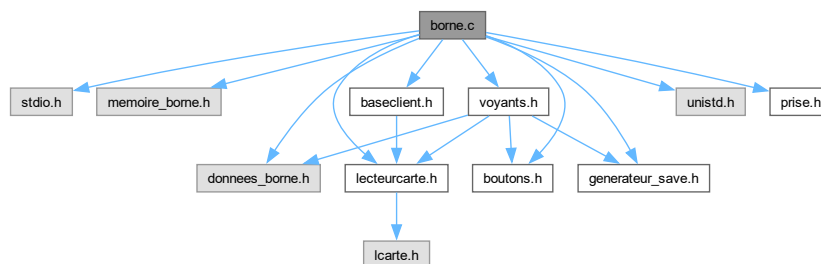
```
00001 #ifndef BASECLIENT_H
00002 #define BASECLIENT_H
00003 #include "lecteurcarte.h"
00004
00005 int base_client_authentification();
00006 void base_client_creer_client();
00007 void base_client_modifier_client();
00008 void base_client_supprimer_client();
00009
00010 #endif // BASECLIENT_H
```

4.5 Référence du fichier borne.c

Programme principal de gestion de la borne de recharge.

```
#include <stdio.h>
#include <memoire_borne.h>
#include <donnees_borne.h>
#include "lecteurcarte.h"
#include "voyants.h"
#include "boutons.h"
#include <unistd.h>
#include "generateur_save.h"
#include "prise.h"
#include "baseclient.h"
```

Graphe des dépendances par inclusion de borne.c:



Fonctions

— int `main` ()

4.5.1 Description détaillée

Programme principal de gestion de la borne de recharge.

Auteur

(Nshuti Nkurikiyinka Fidele, Mapako Eddy)

Version

1.0

Date

25 décembre 2025

Ce programme assure la gestion complète d'une session de recharge :

- Authentification du client via sa carte
- création du client
- Choix des actions (charger, modifier compte, supprimer compte)
- Gestion des voyants et de l'affichage utilisateur
- Détection et gestion du bouton CHARGE
- Séquence complète de charge (DC → AC → AC_CL → fin de charge)
- Gestion sécurisée du STOP avec authentification
- Manipulation de la trappe et du contacteur
- Validation de la carte pour débiter et terminer la charge

Définition dans le fichier `borne.c`.

4.5.2 Documentation des fonctions

4.5.2.1 main()

```
int main ()
```

4.5.3 CLIENT RECONNU

Attente du bouton CHARGE pendant 8 secondes avec clignotement des voyants

DÉBUT PROCÉDURE DE CHARGE

Boucle de surveillance tension + bouton STOP

Gestion AC_CL + Contacteur

Phase retour DC + ouverture trappe

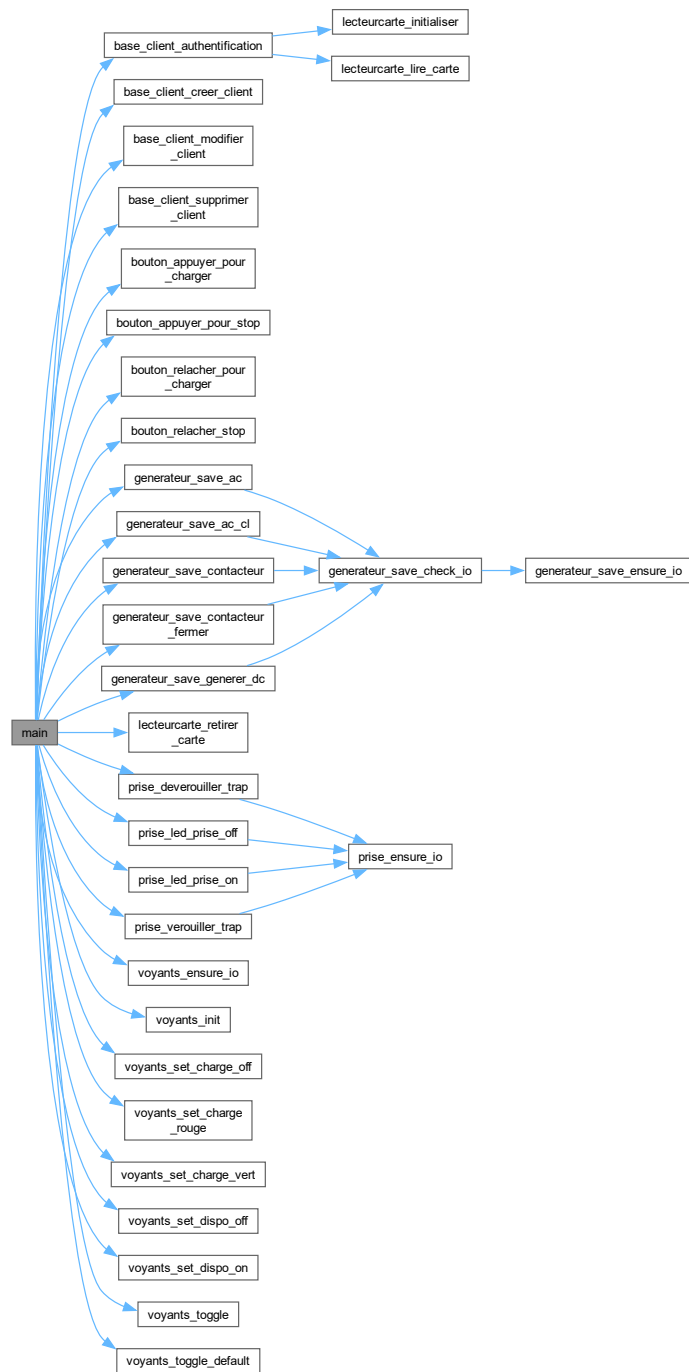
Fin de charge : Vérification carte client

CLIENT NON RECONNU

REMISE À ZÉRO SÉCURISÉE

Définition à la ligne 35 du fichier [borne.c](#).

Voici le graphe d'appel pour cette fonction :



4.6 borne.c

[Aller à la documentation de ce fichier.](#)

```

00001
00019
00020
00021 #include <stdio.h>
00022 #include <memoire_borne.h>
00023 #include <donnees_borne.h>
00024 #include "lecteurcarte.h"

```

```

00025 #include "voyants.h"
00026 #include "boutons.h"
00027 #include <unistd.h>
00028 #include "lecteurcarte.h"
00029 #include "generateur_save.h"
00030 #include "prise.h"
00031 #include "baseclient.h"
00032
00033
00034
00035 int main()
00036 {
00037     /* Variables liées aux états utilisateur et matériel */
00038     int carte;
00039
00040     int charge; //pour mettre le bouton charge a 1
00041     //mettre a zero le stop
00042     int depart_timer;
00043     int retire_carte;
00044     int stop;
00045     int client_charge;
00046     int carte_2=0;
00047     int carte_stop;
00048     volatile int choix_stop = 0;
00049
00050
00051     /* Accès mémoire partagée */
00052     entrees* io = NULL;
00053     int shmid ;
00054
00055     if (io != NULL) return 0;
00056     io = acces_memoire(&shmid);
00057     if (io == NULL) {
00058         fprintf(stderr, "Erreur: acces memoire partagee impossible\n");
00059     }
00060
00061
00062     /* Authentification du client */
00063     carte = base_client_authentification();
00064
00065
00066     //validation = lecteur_carte_valide_carte();
00067
00068     voyants_init();
00069     voyants_ensure_io();
00070
00071     if(carte == 1){
00072         char choix;
00073         printf("Appuyer [C] pour charger le vehicule, [M] pour modifier votre compte, [S] pour
00074 supprimer votre compte\n");
00075         scanf(" %c",&choix);
00076         getchar(); //pour enlever le newline du buffer
00077         switch (choix){
00078             case 'M':
00079                 base_client_modifier_client();
00080                 printf("Au revoir.\n");
00081                 return 0;
00082                 break;
00083             case 'S':
00084                 base_client_supprimer_client();
00085                 printf("Au revoir.\n");
00086                 return 0;
00087                 break;
00088             case 'C':
00089                 printf("Appuyer sur charge et Relaxe vous pendant que le vehicule se
00090 charge...\n");
00091                 client_charge = 1;
00092                 break;
00093             default:
00094                 printf("Choix invalide au revoir.\n");
00095                 return 0;
00096                 break;
00097         }
00098
00099         depart_timer = io->timer_sec;
00100         while(io->timer_sec - depart_timer != 8){
00101             while(1){
00102                 voyants_toggle();//faire clignoter ici
00103                 charge= bouton_appuyer_pour_charger(); //le changement d'etat ne s'opere que q
00104 l'execution suivqnte
00105                 if(charge == 1 || (io->timer_sec - depart_timer) >= 8){
00106                     break;
00107                 }

```

```

00118         }
00119         break;
00120     }
00121 }
00122
00126 if (client_charge == 1){
00127     printf("veuillez s'ils vous plait retirer la carte...\n");
00128
00132 while(1){//pour stop
00133     /* Validation retrait carte */
00134     if (charge == 1 ){
00135         bouton_relacher_pour_charger();
00136         voyants_set_charge_vert(); // ici
00137         voyants_set_dispo_off();
00138         retire_carte = lecteurcarte_retirer_carte();
00139         while (retire_carte != 1);
00140         voyants_set_charge_rouge(); //rouge ici
00141         prise_deverouiller_trap();
00142
00143         generateur_save_generer_dc();
00144         sleep(1);
00145         while (io->gene_u != 9){
00146             sleep(1);
00147             stop = bouton_appuyer_pour_stop();
00148             if(io->gene_u == 9){
00149                 prise_led_prise_on();
00150                 prise_verouiller_trap();
00151                 generateur_save_ac();
00152                 sleep(1);
00153                 break;
00154             }
00155             else if (stop == 1){
00156                 carte_stop = base_client_authentication();
00157                 if (carte_stop == 1){
00158
00159                     choix_stop = 1;
00160                     sleep(0.5);
00161                     break;
00162                 }
00163             }
00164         }
00165     }
00166     if (stop == 1 ){
00167
00168         if (choix_stop == 1){
00169             //sleep(1);
00170             break;
00171         }
00172         else if (choix_stop != 1) {
00173             carte_stop = base_client_authentication();
00174             if (carte_stop == 1){
00175                 sleep(1);
00176                 choix_stop = 1;
00177                 break;
00178             }
00179         }
00180     }
00181 }
00182 }
00183 }
00184 }
00185 stop = bouton_appuyer_pour_stop();
00186 if(io->gene_u == 6 && stop == 0){
00187     generateur_save_ac_cl();
00188     generateur_save_contacteur();
00189     sleep(0.5);
00190 }
00191 else if (stop == 1 ){
00192     if (choix_stop == 1){
00193         //sleep(1);
00194         break;
00195     }
00196     else if (choix_stop != 1) {
00197         carte_stop = base_client_authentication();
00198         if (carte_stop == 1){
00199             sleep(1);
00200             choix_stop = 1;
00201             break;
00202         }
00203     }
00204 }
00205 }
00206 }
00207 }
00208 }
00212 while(1){
00213     //sleep(1);
00214     stop = bouton_appuyer_pour_stop();
00215     if (io->gene_u == 9 && stop == 0){
00216         sleep(0.5);

```

```

00217
00218         generateur_save_contacteur_fermer();
00219         // }
00220         generateur_save_generer_dc();
00221         //prise_deverouiller_trap();
00222         voyants_set_charge_vert();// vert ici
00223
00224         break;
00225     }
00226     sleep(0.5);
00227     if (stop == 1){
00228         if (choix_stop == 1){
00229             //sleep(1);
00230             // break;
00231             continue;
00232         }
00233         else if (choix_stop != 1) {
00234             carte_stop = base_client_authentication();
00235             if (carte_stop == 1){
00236                 sleep(1);
00237                 choix_stop = 1;
00238                 //break;
00239             }
00240
00241             break;
00242         }
00243         break;
00244     }
00245 }
00246
00247 }
00251     sleep(1);
00252     if(choix_stop != 1){
00253         carte_2 = base_client_authentication();
00254     }
00255     while (1){
00256
00257         while (io->gene_u != 12 && stop == 0){
00258             sleep(1);
00259             stop = bouton_appuyer_pour_stop();
00260             if(carte_2 == 1){
00261                 prise_deverouiller_trap();
00262                 sleep(0.5);
00263             }
00264             if(io->gene_u == 12 && stop == 0 && carte_2 == 1){
00265                 sleep(0.5);
00266                 voyants_set_charge_off();//OFF ici
00267                 prise_led_prise_off();
00268                 prise_verouiller_trap();
00269
00270
00271                 break;
00272             }
00273             else if(carte_2 != 1){
00274                 printf("veuillez s'il vous plait utilisez la bonne carte\n");
00275                 sleep(1);
00276                 carte_2 = base_client_authentication();
00277             }
00278             else if (stop == 1){
00279
00280                 if (choix_stop == 1){
00281                     //sleep(1);
00282                     break;
00283                 }
00284                 else if (choix_stop != 1) {
00285                     carte_stop = base_client_authentication();
00286
00287                     if (carte_stop == 1){
00288                         sleep(1);
00289                         choix_stop = 1;
00290                         break;
00291                     }
00292                     else{
00293                         continue;
00294                     }
00295
00296                 }
00297             }
00298         }
00299     }
00300     break;
00301 }
00302
00303 // break;
00304
00305 //printf("je suis ici 2\n");
00306

```

```

00307
00308     }
00309     else if(io->timer_sec - depart_timer >= 8){
00310         bouton_relacher_pour_charger();
00311         voyants_set_charge_off();// off ici
00312         voyants_set_dispo_on();
00313         break;
00314     }
00315 }
00316 if (choix_stop == 1){
00317     prise_deverouiller_trap();
00318     while (io->gene_u == 0 || io->gene_u == 6|| io->gene_u == 9 || io->gene_u == 12){
00319         sleep(1);
00320         // if(io->gene_u == 12){
00321             //sleep(1);
00322             //printf("je suis ici 1\n");
00323             voyants_set_charge_off();//OFF ici
00324             prise_led_prise_off();
00325             generateur_save_contacteur_fermer();
00326             //prise_verouiller_trap();
00327             sleep(5);
00328             choix_stop = 0;
00329             bouton_relacher_stop();
00330
00331
00332             break;
00333         // }
00334     }
00335     break;
00336 }
00337 }
00338 else{
00339     break;
00340 }
00341 }
00342 }
00343 else if (client_charge != 1) {
00344     printf("vous n'avez pas appuyer, veuillez relancer la borne\n");
00345     voyants_init();
00346     bouton_relacher_stop();
00347     bouton_relacher_pour_charger();
00348     generateur_save_contacteur_fermer();
00349 }
00350 }
00351 }
00352 else {
00353     char choix_creation;
00354     depart_timer = io->timer_sec;
00355     printf("veuillez attendre la fin du clignotement\n");
00356     while(io->timer_sec - depart_timer != 8){
00357         while(1){
00358             voyants_toggle_default();//faire clignoter ici
00359             if((io->timer_sec - depart_timer) >= 8){
00360                 break;
00361             }
00362         }
00363         break;
00364     }
00365     printf("Client non reconnu, voulez vous creez un compte [Y] [N]...\n");
00366     scanf(" %c",&choix_creation);
00367
00368     getchar(); //pour enlever le newline du buffer
00369     if(choix_creation == 'Y' || choix_creation == 'y'){
00370         base_client_creer_client();
00371     }
00372     else{
00373         printf("Au revoir !\n");
00374     }
00375 }
00376 }
00377 }
00378 }
00379 }
00380 }
00381 if (choix_stop == 1){
00382     prise_deverouiller_trap();
00383     while (io->gene_u == 0 || io->gene_u == 6|| io->gene_u == 9 || io->gene_u == 12){
00384         //sleep(1);
00385         // if(io->gene_u == 12){
00386             sleep(1);
00387             //printf("je suis ici 1\n");
00388             voyants_set_charge_off();//OFF ici
00389             prise_led_prise_off();
00390             generateur_save_contacteur_fermer();
00391             //prise_verouiller_trap();
00392             sleep(5);
00393
00394             break;
00395         // }
00396     }

```

```

00397     }
00398 }
00399
00403     //generateur_save_contacteur_fermer();
00404     prise_verouiller_trap();
00405     voyants_init();
00406     bouton_relacher_stop();
00407     bouton_relacher_pour_charger();
00408     generateur_save_contacteur_fermer();
00409     prise_verouiller_trap();
00410
00411
00412
00413
00414     return 0;
00415 }
00416 }
00417

```

4.7 Référence du fichier boutons.c

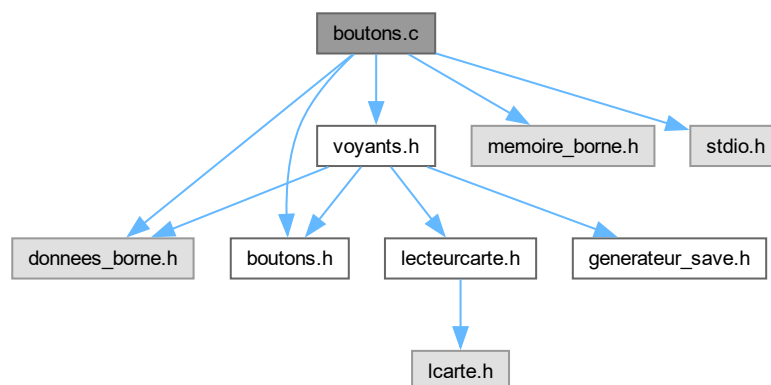
programmes pour gerer les boutons sur la borne.

```

#include <donnees_borne.h>
#include <memoire_borne.h>
#include "voyants.h"
#include <stdio.h>
#include "boutons.h"

```

Graphe des dépendances par inclusion de boutons.c:



Fonctions

- void bouton_ensure_io ()
fonction pour s'assurer que la connexion a la memoire partagee est etabli.
- int bouton_appuyer_pour_charger ()
fonction pour mettre a 1 le bouton charge.
- int bouton_relacher_pour_charger ()
fonction pour mettre a 0 le bouton charge.
- int bouton_appuyer_pour_stop ()
fonction pour mettre a 1 le bouton stop.
- int bouton_relacher_stop ()
fonction pour mettre a 0 le bouton stop.

Variables

- entrees * io_b

— int `shmid_b`
Identifiant de la mémoire partagée.

4.7.1 Description détaillée

programmes pour gerer les boutons sur la borne.

Auteur

(Nshuti Nkurikiyinka Fidele, Mapako Eddy)

Version

1.0

Date

22 decembre 2025

programmes pour gerer l'appuie et le relachement du bouton charge et stop
Définition dans le fichier [boutons.c](#).

4.7.2 Documentation des fonctions

4.7.2.1 bouton_appuyer_pour_charger()

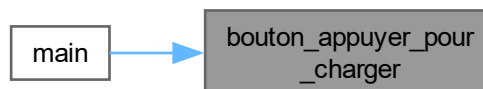
`int bouton_appuyer_pour_charger ()`
fonction pour mettre a 1 le bouton charge.

Renvoie

1 quand le bouton charge est appuye

Définition à la ligne 64 du fichier [boutons.c](#).

Voici le graphe des appelants de cette fonction :



4.7.2.2 bouton_appuyer_pour_stop()

`int bouton_appuyer_pour_stop ()`
fonction pour mettre a 1 le bouton stop.

Renvoie

1 quand le bouton stop est appuyé

Définition à la ligne 95 du fichier [boutons.c](#).

Voici le graphe des appelants de cette fonction :

**4.7.2.3 bouton_ensure_io()**

```
void bouton_ensure_io ()
```

fonction pour s'assurer que la connexion à la mémoire partagée est établie.

Renvoie

si la connexion est déjà établie préalablement

Définition à la ligne 37 du fichier [boutons.c](#).

4.7.2.4 bouton_relacher_pour_charger()

```
int bouton_relacher_pour_charger ()
```

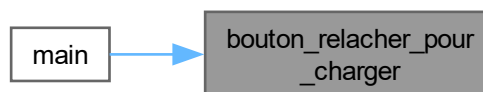
fonction pour mettre à 0 le bouton chargé.

Renvoie

0 quand la fonction est appelée

Définition à la ligne 78 du fichier [boutons.c](#).

Voici le graphe des appelants de cette fonction :

**4.7.2.5 bouton_relacher_stop()**

```
int bouton_relacher_stop ()
```

fonction pour mettre à 0 le bouton stop.

Renvoie

0 quand la fonction est appelée

Définition à la ligne 110 du fichier [boutons.c](#).

Voici le graphe des appelants de cette fonction :



4.7.3 Documentation des variables

4.7.3.1 io_b

entrees* io_b

Définition à la ligne 23 du fichier [boutons.c](#).

4.7.3.2 shmid_b

int shmid_b

Identifiant de la mémoire partagée.

Définition à la ligne 29 du fichier [boutons.c](#).

4.8 boutons.c

[Aller à la documentation de ce fichier.](#)

```

00001
00011
00012 #include <donnees_borne.h>
00013 #include <memoire_borne.h>
00014 #include "voyants.h"
00015 #include <stdio.h>
00016 #include "boutons.h"
00017
00018
00023 entrees* io_b;
00024
00029 int shmid_b;
00030
00037 void bouton_ensure_io()
00038 {
00039     if (io_b != NULL) return;
00040     io_b = acces_memoire(&shmid_b);
00041     if (io_b == NULL) {
00042         fprintf(stderr, "Erreur: acces memoire partagee impossible\n");
00043     }
00044 }
00045
00052 static int bouton_check_io()
00053 {
00054     if (io_b == NULL) bouton_ensure_io();
00055     return (io_b != NULL);
00056 }
00057
00064 int bouton_appuyer_pour_charger() {
00065     int en_charge;
00066     if (!bouton_check_io()) return 0; //erreur de sigma
00067     //io_b->bouton_charge;
00068     en_charge = io_b->bouton_charge;
00069     return en_charge;
00070 }
00071
00078 int bouton_relacher_pour_charger() {
  
```

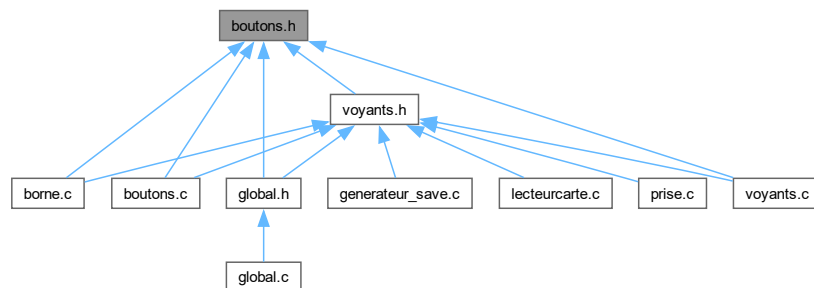
```

00079     int charge;
00080     if (!bouton_check_io()) return 0; //erreur de segma
00081     io_b->bouton_charge = 0;
00082     charge = io_b ->bouton_charge;
00083     //printf("%d \n", charge);
00084
00085     return charge;
00086 }
00087
00088
00095 int bouton_appuyer_pour_stop() {
00096     int stop;
00097     if (!bouton_check_io()) return 0; //erreur de segma
00098     //io_b->bouton_stop;
00099     stop = io_b ->bouton_stop;
00100     return stop;
00101 }
00102
00103
00110 int bouton_relacher_stop() {
00111     int stop;
00112     if (!bouton_check_io()) return 0; //erreur de segma
00113     io_b->bouton_stop = 0;
00114     stop = io_b ->bouton_stop;
00115     return stop;
00116 }
00117 }

```

4.9 Référence du fichier boutons.h

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Fonctions

- int bouton_appuyer_pour_charger ()
fonction pour mettre a 1 le bouton charge.
- void bouton_ensure_io ()
fonction pour s'assurer que la connexion a la memoire partagee est etabli.
- int bouton_relacher_pour_charger ()
fonction pour mettre a 0 le bouton charge.
- int bouton_appuyer_pour_stop ()
fonction pour mettre a 1 le bouton stop.
- int bouton_relacher_stop ()
fonction pour mettre a 0 le bouton stop.

4.9.1 Documentation des fonctions

4.9.1.1 bouton_appuyer_pour_charger()

```
int bouton_appuyer_pour_charger ()
```

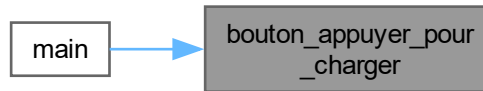
fonction pour mettre a 1 le bouton charge.

Renvoie

1 quand le bouton charge est appuye

Définition à la ligne 64 du fichier [boutons.c](#).

Voici le graphe des appelants de cette fonction :

**4.9.1.2 bouton_appuyer_pour_stop()**

```
int bouton_appuyer_pour_stop ()
```

fonction pour mettre a 1 le bouton stop.

Renvoie

1 quand le bouton stop est appuye

Définition à la ligne 95 du fichier [boutons.c](#).

Voici le graphe des appelants de cette fonction :

**4.9.1.3 bouton_ensure_io()**

```
void bouton_ensure_io ()
```

fonction pour s'assurer que la connexion a la memoire partagee est etabli.

Renvoie

si la connexion est deja etabli prealablement

Définition à la ligne 37 du fichier [boutons.c](#).

4.9.1.4 bouton_relacher_pour_charger()

```
int bouton_relacher_pour_charger ()
```

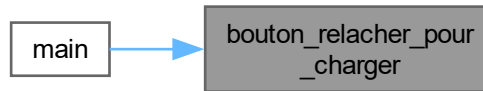
fonction pour mettre a 0 le bouton charge.

Renvoie

0 quand la fonction est appelée

Définition à la ligne 78 du fichier [boutons.c](#).

Voici le graphe des appelants de cette fonction :

**4.9.1.5 bouton_relacher_stop()**

```
int bouton_relacher_stop ()
```

fonction pour mettre à 0 le bouton stop.

Renvoie

0 quand la fonction est appelée

Définition à la ligne 110 du fichier [boutons.c](#).

Voici le graphe des appelants de cette fonction :

**4.10 boutons.h**

[Aller à la documentation de ce fichier.](#)

```

00001 #ifndef BOUTONS_H
00002 #define BOUTONS_H
00003 int bouton_appuyer_pour_charger();
00004 void bouton_ensure_io();
00005 int bouton_relacher_pour_charger();
00006 int bouton_appuyer_pour_stop();
00007 int bouton_relacher_stop();
00008 #endif
  
```

4.11 Référence du fichier generateur_save.c

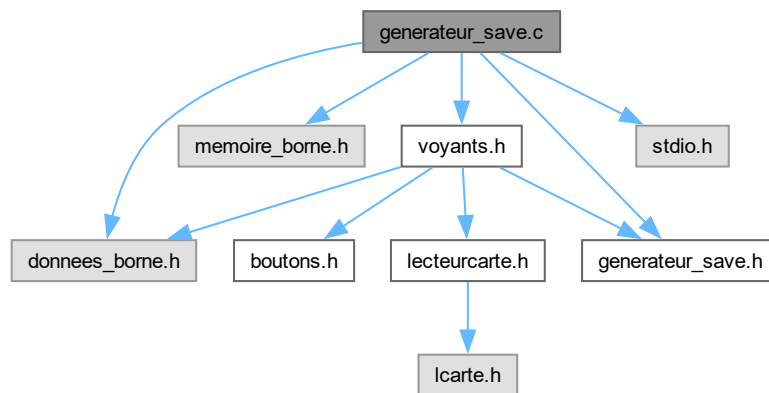
Programme pour gérer le générateur et le contacteur via la mémoire partagée.

```

#include <donnees_borne.h>
#include <memoire_borne.h>
#include "voyants.h"
#include <stdio.h>
  
```

```
#include "generateur_save.h"
```

Graphe des dépendances par inclusion de generateur_save.c:



Fonctions

- void [generateur_save_ensure_io](#) ()
Fonction pour s'assurer que la connexion à la mémoire partagée est établie.
- int [generateur_save_check_io](#) ()
Vérifie si la mémoire partagée est accessible et l'établit si nécessaire.
- void [generateur_save_generer_dc](#) ()
genere 12v DC.
- void [generateur_save_ac](#) ()
Genere AC de 1KHz.
- void [generateur_save_ac_cl](#) ()
Generer PWM AC variable.
- void [generateur_save_contacteur](#) ()
Active le contacteur AC (fermeture du relais).
- void [generateur_save_contacteur_fermer](#) ()
Désactive le contacteur AC (ouverture du relais).

Variables

- entrees * [io_g](#)
Pointeur vers la structure des entrées partagées.
- int [shmid_g](#)
Identifiant de la mémoire partagée.

4.11.1 Description détaillée

Programme pour gérer le générateur et le contacteur via la mémoire partagée.

Auteur

(Nshuti Nkurikiyinka Fidele, Mapako Eddy)

Version

1.0

Date

25 décembre 2025

Ce fichier permet d'accéder à la mémoire partagée de la borne et de piloter :

- Le générateur PWM (DC, AC 1kHz, AC CL)
- Le contacteur AC

Définition dans le fichier [generateur_save.c](#).

4.11.2 Documentation des fonctions

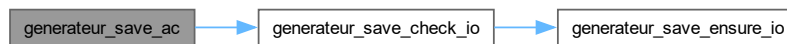
4.11.2.1 `generateur_save_ac()`

```
void generateur_save_ac ()
```

Genere AC de 1KHz.

Définition à la ligne 74 du fichier [generateur_save.c](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



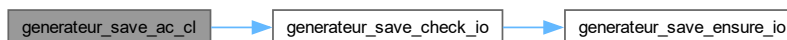
4.11.2.2 `generateur_save_ac_cl()`

```
void generateur_save_ac_cl ()
```

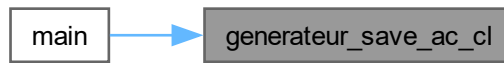
Generer PWM AC variable.

Définition à la ligne 84 du fichier [generateur_save.c](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



4.11.2.3 generateur_save_check_io()

```
int generateur_save_check_io ()
```

Vérifie si la mémoire partagée est accessible et l'établit si nécessaire.

Renvoie

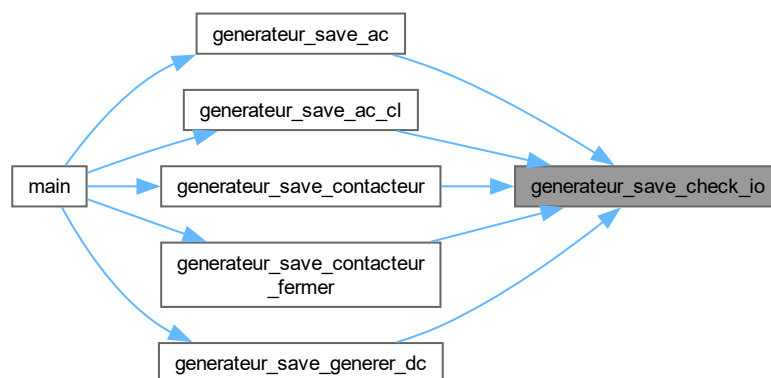
1 si l'accès est valide, 0 sinon.

Définition à la ligne 52 du fichier [generateur_save.c](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



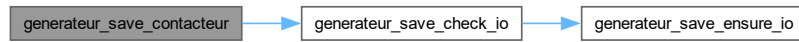
4.11.2.4 generateur_save_contacteur()

```
void generateur_save_contacteur ()
```

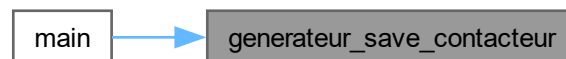
Active le contacteur AC (fermeture du relais).

Définition à la ligne 94 du fichier `generateur_save.c`.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



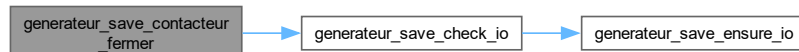
4.11.2.5 `generateur_save_contacteur_fermer()`

```
void generateur_save_contacteur_fermer ()
```

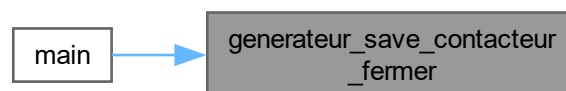
Désactive le contacteur AC (ouverture du relais).

Définition à la ligne 104 du fichier `generateur_save.c`.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



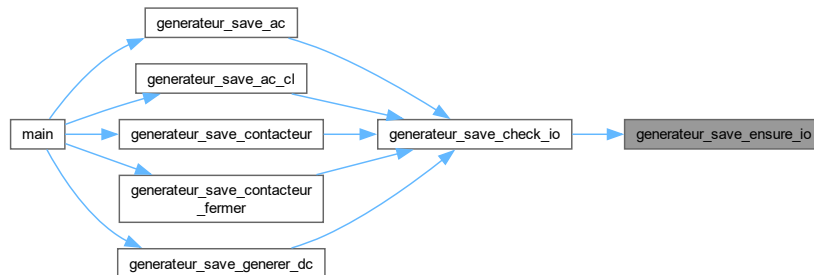
4.11.2.6 `generateur_save_ensure_io()`

```
void generateur_save_ensure_io ()
```

Fonction pour s'assurer que la connexion à la mémoire partagée est établie.

Établit l'accès à la mémoire partagée si ce n'est pas déjà fait. Affiche un message d'erreur si la connexion est impossible.

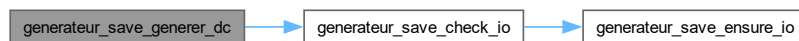
Définition à la ligne 38 du fichier [generateur_save.c](#).
Voici le graphe des appelants de cette fonction :



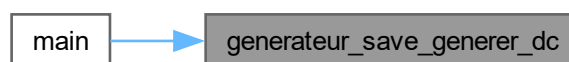
4.11.2.7 generateur_save_generer_dc()

```
void generateur_save_generer_dc ()
```

genere 12v DC.
Définition à la ligne 62 du fichier [generateur_save.c](#).
Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



4.11.3 Documentation des variables

4.11.3.1 io_g

```
entrees* io_g
```

Pointeur vers la structure des entrées partagées.
Définition à la ligne 24 du fichier [generateur_save.c](#).

4.11.3.2 shmid_g

```
int shmid_g
```

Identifiant de la mémoire partagée.
Définition à la ligne 29 du fichier [generateur_save.c](#).

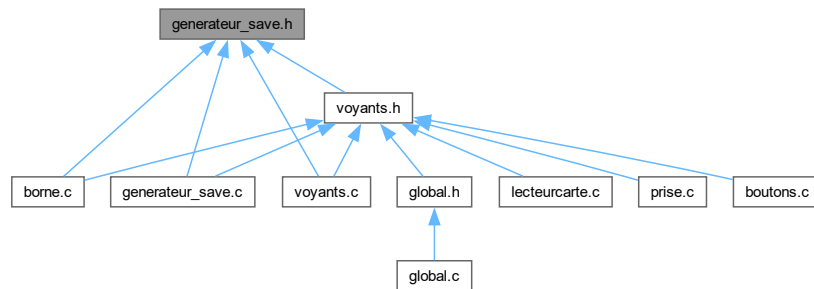
4.12 generateur_save.c

[Aller à la documentation de ce fichier.](#)

```
00001
00012
00013
00014 #include <donnees_borne.h>
00015 #include <memoire_borne.h>
00016 #include "voyants.h"
00017 #include <stdio.h>
00018 #include "generateur_save.h"
00019
00024 entrees* io_g;
00029 int shmid_g;
00030
00038 void generateur_save_ensure_io()
00039 {
00040     if (io_g != NULL) return;
00041     io_g = acces_memoire(&shmid_g);
00042     if (io_g == NULL) {
00043         fprintf(stderr, "Erreur: acces memoire partagee impossible\n");
00044     }
00045 }
00046
00052 int generateur_save_check_io()
00053 {
00054     if (io_g == NULL) generateur_save_ensure_io();
00055     return (io_g != NULL);
00056 }
00057
00062 void generateur_save_generer_dc() {
00063     if (!generateur_save_check_io()) return;
00064
00065     io_g ->gene_pwm = DC;
00067 }
00068
00073
00074 void generateur_save_ac() {
00075     if (!generateur_save_check_io()) return;
00076
00077     io_g ->gene_pwm = AC_1K;
00078 }
00079
00084 void generateur_save_ac_cl() {
00085     if (!generateur_save_check_io()) return;
00086
00087     io_g ->gene_pwm = AC_CL;
00088 }
00089
00094 void generateur_save_contacteur() {
00095     if (!generateur_save_check_io()) return;
00096
00097     io_g->contacteur_AC = 1;
00098 }
00099
00104 void generateur_save_contacteur_fermer() {
00105     if (!generateur_save_check_io()) return;
00106
00107     io_g->contacteur_AC = 0;
00108 }
00109
00110
00111
```

4.13 Référence du fichier `generateur_save.h`

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Fonctions

- int `generateur_save_check_io` ()
Vérifie si la mémoire partagée est accessible et l'établit si nécessaire.
- void `generateur_save_ensure_io` ()
Fonction pour s'assurer que la connexion à la mémoire partagée est établie.
- void `generateur_save_generer_dc` ()
genere 12v DC.
- void `generateur_save_ac` ()
Genere AC de 1KHz.
- void `generateur_save_ac_cl` ()
Generer PWM AC variable.
- void `generateur_save_contacteur` ()
Active le contacteur AC (fermeture du relais).
- void `generateur_save_contacteur_fermer` ()
Désactive le contacteur AC (ouverture du relais).

4.13.1 Documentation des fonctions

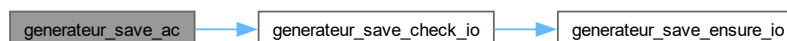
4.13.1.1 `generateur_save_ac()`

```
void generateur_save_ac ()
```

Genere AC de 1KHz.

Définition à la ligne 74 du fichier `generateur_save.c`.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



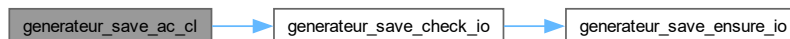
4.13.1.2 `generateur_save_ac_cl()`

```
void generateur_save_ac_cl ()
```

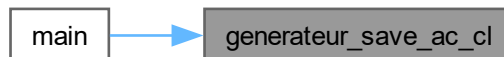
Générer PWM AC variable.

Définition à la ligne 84 du fichier [generateur_save.c](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



4.13.1.3 `generateur_save_check_io()`

```
int generateur_save_check_io ()
```

Vérifie si la mémoire partagée est accessible et l'établit si nécessaire.

Renvoie

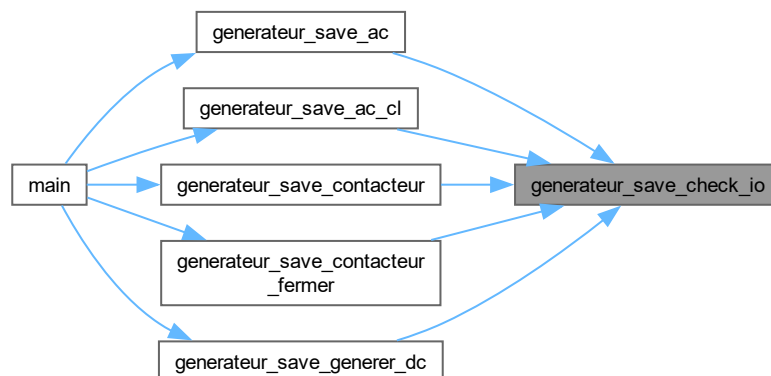
1 si l'accès est valide, 0 sinon.

Définition à la ligne 52 du fichier [generateur_save.c](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



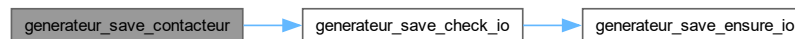
4.13.1.4 `generateur_save_contacteur()`

```
void generateur_save_contacteur ()
```

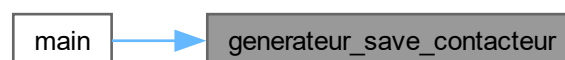
Active le contacteur AC (fermeture du relais).

Définition à la ligne 94 du fichier [generateur_save.c](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



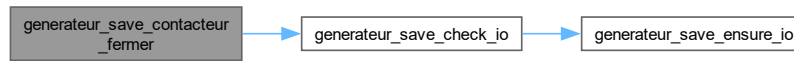
4.13.1.5 `generateur_save_contacteur_fermer()`

```
void generateur_save_contacteur_fermer ()
```

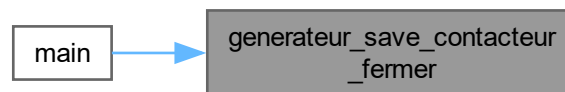
Désactive le contacteur AC (ouverture du relais).

Définition à la ligne 104 du fichier [generateur_save.c](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



4.13.1.6 `generateur_save_ensure_io()`

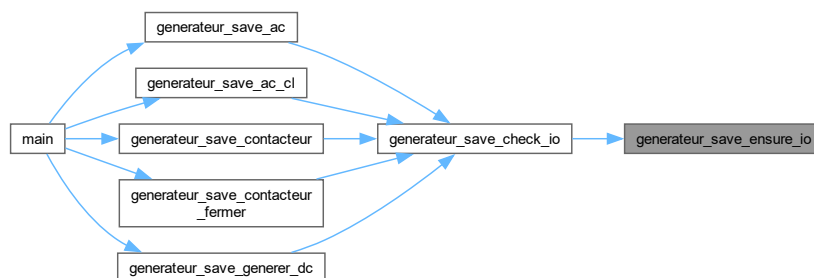
```
void generateur_save_ensure_io ()
```

Fonction pour s'assurer que la connexion à la mémoire partagée est établie.

Établit l'accès à la mémoire partagée si ce n'est pas déjà fait. Affiche un message d'erreur si la connexion est impossible.

Définition à la ligne 38 du fichier `generateur_save.c`.

Voici le graphe des appelants de cette fonction :



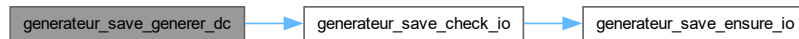
4.13.1.7 `generateur_save_generer_dc()`

```
void generateur_save_generer_dc ()
```

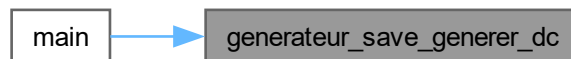
genere 12v DC.

Définition à la ligne 62 du fichier `generateur_save.c`.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



4.14 generateur_save.h

[Aller à la documentation de ce fichier.](#)

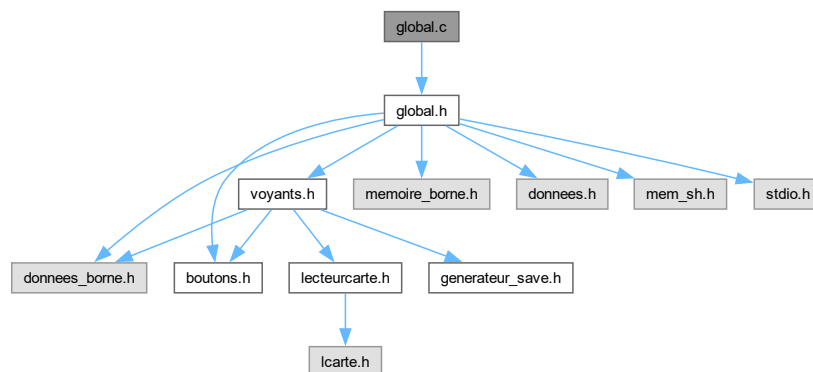
```

00001 #ifndef GENERATEUR_SAVE_H
00002 #define GENERATEUR_SAVE_H
00003
00004 int generateur_save_check_io();
00005 void generateur_save_ensure_io();
00006 void generateur_save_generer_dc();
00007 void generateur_save_ac();
00008 void generateur_save_ac_cl();
00009 void generateur_save_contacteur();
00010 void generateur_save_contacteur_fermer();
00011 #endif
  
```

4.15 Référence du fichier global.c

```
#include "global.h"
```

Graphe des dépendances par inclusion de global.c:



Variables

- entrees* `io_b` = NULL
- int `shmid_b` = 0

4.15.1 Documentation des variables

4.15.1.1 `io_b`

```
entrees* io_b = NULL
```

Définition à la ligne 2 du fichier `global.c`.

4.15.1.2 `shmid_b`

```
int shmid_b = 0
```

Définition à la ligne 3 du fichier `global.c`.

4.16 global.c

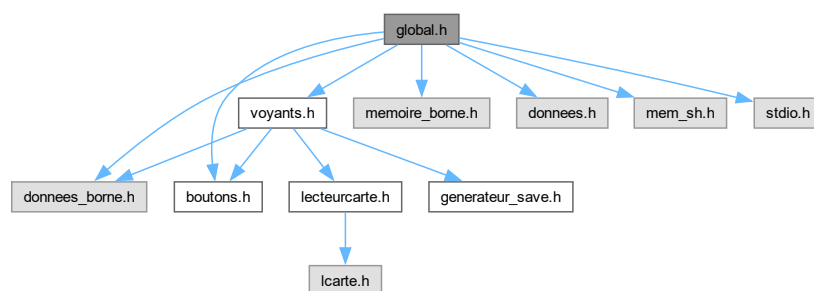
[Aller à la documentation de ce fichier.](#)

```
00001 #include "global.h"
00002 entrees* io_b = NULL;
00003 int shmid_b = 0;
```

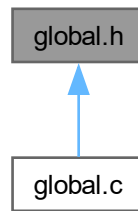
4.17 Référence du fichier global.h

```
#include <donnees_borne.h>
#include <memoire_borne.h>
#include "voyants.h"
#include <donnees.h>
#include <mem_sh.h>
#include <stdio.h>
#include "boutons.h"
```

Graphe des dépendances par inclusion de `global.h`:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Variables

- entrees * [io_b](#)
- int [shmid_b](#)

4.17.1 Documentation des variables

4.17.1.1 io_b

entrees* [io_b](#)

Définition à la ligne 10 du fichier [global.h](#).

4.17.1.2 shmid_b

int [shmid_b](#)

Définition à la ligne 11 du fichier [global.h](#).

4.18 global.h

[Aller à la documentation de ce fichier.](#)

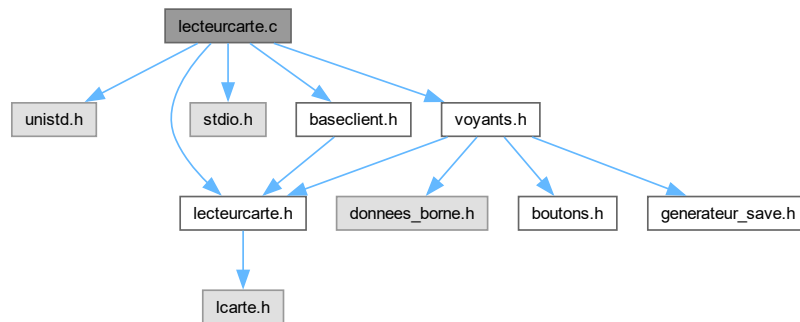
```
00001 #ifndef VOYANTS_H
00002 #define VOYANTS_H
00003 #include <donnees_borne.h>
00004 #include <memoire_borne.h>
00005 #include "voyants.h"
00006 #include <donnees.h>
00007 #include <mem_sh.h>
00008 #include <stdio.h>
00009 #include "boutons.h"
00010 entrees* io\_b;
00011 int shmid\_b ;
00012 #endif
```

4.19 Référence du fichier lecteurcarte.c

Gestion du lecteur de carte utilisateur.

```
#include <unistd.h>
#include "lecteurcarte.h"
#include <stdio.h>
#include "baseclient.h"
#include "voyants.h"
```

Graphe des dépendances par inclusion de lecteurcarte.c:



Fonctions

- void `lecteurcarte_initialiser()`
Initialise le lecteur de carte.
- int `lecteurcarte_lire_carte()`
Attend l'insertion d'une carte et lit le numéro de badge.
- int `lecteurcarte_retirer_carte()`
Attend le retrait de la carte par l'utilisateur.

Variables

- volatile int `numero`

4.19.1 Description détaillée

Gestion du lecteur de carte utilisateur.

Auteur

(Nshuti Nkurikiyinka Fidele, Mapako Eddy)

Version

1.0

Date

25 décembre 2025

Ce module gère :

- l'initialisation du lecteur de carte
- la lecture du numéro de badge utilisateur
- l'attente du retrait de la carte

Définition dans le fichier `lecteurcarte.c`.

4.19.2 Documentation des fonctions

4.19.2.1 `lecteurcarte_initialiser()`

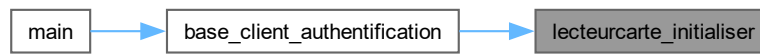
```
void lecteurcarte_initialiser ()
```

Initialise le lecteur de carte.

Prépare le lecteur de carte pour permettre la lecture des badges utilisateurs.

Définition à la ligne 31 du fichier `lecteurcarte.c`.

Voici le graphe des appelants de cette fonction :



4.19.2.2 `lecteurcarte_lire_carte()`

```
int lecteurcarte_lire_carte ()
```

Attend l'insertion d'une carte et lit le numéro de badge.

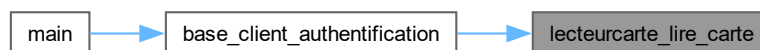
Affiche une demande à l'utilisateur, attend l'insertion, puis récupère le numéro de carte.

Renvoie

le numéro du badge lu.

Définition à la ligne 46 du fichier [lecteurcarte.c](#).

Voici le graphe des appelants de cette fonction :



4.19.2.3 `lecteurcarte_retirer_carte()`

```
int lecteurcarte_retirer_carte ()
```

Attend le retrait de la carte par l'utilisateur.

Bloque l'exécution tant que la carte n'est pas retirée.

Renvoie

1 lorsque la carte a bien été retirée.

Définition à la ligne 69 du fichier [lecteurcarte.c](#).

Voici le graphe des appelants de cette fonction :



4.19.3 Documentation des variables

4.19.3.1 numero

volatile int numero

Définition à la ligne 21 du fichier [lecteurcarte.c](#).

4.20 lecteurcarte.c

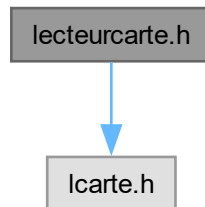
[Aller à la documentation de ce fichier.](#)

```
00001
00014
00015 #include <unistd.h>
00016 #include "lecteurcarte.h"
00017 #include <stdio.h>
00018 #include "baseclient.h"
00019 #include "voyants.h"
00020
00021 volatile int numero;
00022
00030
00031 void lecteurcarte_initialiser()
00032 {
00033     initialisations_ports();
00034 }
00035 }
00036
00046 int lecteurcarte_lire_carte()
00047 {
00048
00049     //lire carte
00050     printf("veuillez sil vous plait rentrez le numero: \n");
00051     attente_insertion_carte();
00052     numero = lecture_numero_carte();
00053     printf("le numero de badge: %d \n",numero);
00054
00055     //liberez carte
00056
00057     return numero;
00058 }
00059 }
00060
00069 int lecteurcarte_retirer_carte()
00070 {
00071     attente_retrait_carte();
00072     //liberation_ports();
00073     return 1;
00074 }
00075 }
00076
00077 /*int lecteur_carte_valide_carte(){
00078 //pas de parametres
00079 //demande n de badge
00080 //retourne 1 si la carte est valide 0 sinon
00081 //appel authentifier
00082 //appel voyants pour voir etat_dispo
00083     int authentication = 1;
00084     //authentication = base_clients_authentifier(numero);
00085     if (authentication == 1){
00086         printf ("success \n");
00087     }
00088     else if (authentication == 100000) //nombre aleatoire
00089     {
00090         printf ("client non trouve");
00091     }
00092 }
00093 }
00094
00095     return authentication;
00096 }*/
00097
00098
00099
```

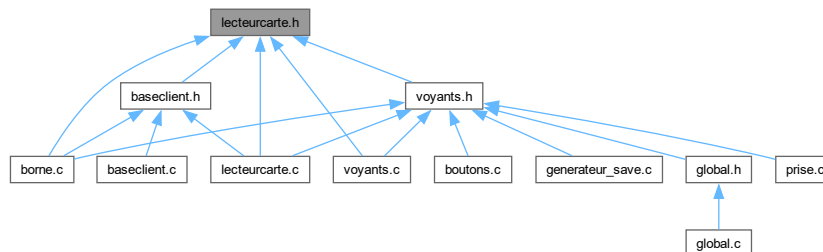
4.21 Référence du fichier lecteurcarte.h

```
#include <lcarte.h>
```

Graphe des dépendances par inclusion de lecteurcarte.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Fonctions

- void `lecteurcarte_initialiser()`
Initialise le lecteur de carte.
- int `lecteurcarte_lire_carte()`
Attend l'insertion d'une carte et lit le numéro de badge.
- int `lecteur_carte_valide_carte()`
- void `lecteur_carte_demande_creation_client()`
- void `lecteur_carte_demande_modifier_client()`
- void `lecteur_carte_demande_supprimer_client()`
- int `lecteurcarte_retirer_carte()`
Attend le retrait de la carte par l'utilisateur.

4.21.1 Documentation des fonctions

4.21.1.1 lecteur_carte_demande_creation_client()

```
void lecteur_carte_demande_creation_client ()
```

4.21.1.2 lecteur_carte_demande_modifier_client()

```
void lecteur_carte_demande_modifier_client ()
```

4.21.1.3 lecteur_carte_demande_supprimer_client()

```
void lecteur_carte_demande_supprimer_client ()
```

4.21.1.4 lecteur_carte_valide_carte()

```
int lecteur_carte_valide_carte ()
```

4.21.1.5 lecteurcarte_initialiser()

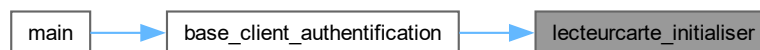
```
void lecteurcarte_initialiser ()
```

Initialise le lecteur de carte.

Prépare le lecteur de carte pour permettre la lecture des badges utilisateurs.

Définition à la ligne 31 du fichier [lecteurcarte.c](#).

Voici le graphe des appelants de cette fonction :



4.21.1.6 lecteurcarte_lire_carte()

```
int lecteurcarte_lire_carte ()
```

Attend l'insertion d'une carte et lit le numéro de badge.

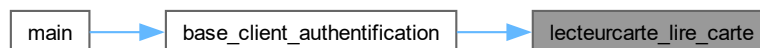
Affiche une demande à l'utilisateur, attend l'insertion, puis récupère le numéro de carte.

Renvoie

le numéro du badge lu.

Définition à la ligne 46 du fichier [lecteurcarte.c](#).

Voici le graphe des appelants de cette fonction :



4.21.1.7 lecteurcarte_retirer_carte()

```
int lecteurcarte_retirer_carte ()
```

Attend le retrait de la carte par l'utilisateur.

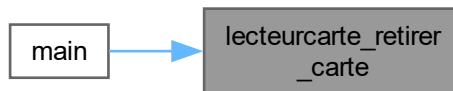
Bloque l'exécution tant que la carte n'est pas retirée.

Renvoie

1 lorsque la carte a bien été retirée.

Définition à la ligne 69 du fichier `lecteurcarte.c`.

Voici le graphe des appelants de cette fonction :



4.22 lecteurcarte.h

[Aller à la documentation de ce fichier.](#)

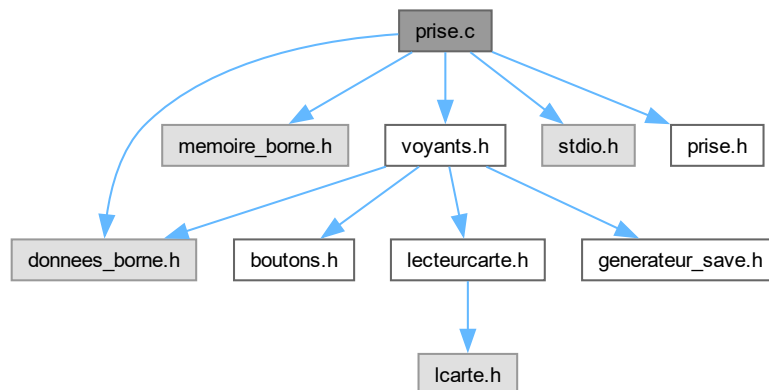
```
00001 #ifndef LECTEURCARTE_H
00002 #define LECTEURCARTE_H
00003 #include <lcarte.h>
00004
00005 void lecteurcarte_initialiser();
00006 int lecteurcarte_lire_carte();
00007 int lecteur_carte_valide_carte();
00008 void lecteur_carte_demande_creation_client();//ne prends rien en parametre, appel la fonction creer
      client et lui passe le nom et prenom
00009 void lecteur_carte_demande_modifier_client();//ne prends rien en parametre, appel la fonction modifier
      client et lui passe le numero de badge
00010 void lecteur_carte_demande_supprimer_client();//ne prends rien en parametre, appel la fonction
      supprimer client et lui passe le numero de badge
00011 int lecteurcarte_retirer_carte();
00012 #endif // LECTEURCARTE_H
```

4.23 Référence du fichier prise.c

Gestion de la prise et de la trappe de la borne.

```
#include <donnees_borne.h>
#include <memoire_borne.h>
#include "voyants.h"
#include <stdio.h>
#include "prise.h"
```

Graphe des dépendances par inclusion de prise.c:



Fonctions

- void `prise_ensure_io` ()
Initialise l'accès à la mémoire partagée si nécessaire.
- int `prise_check_io` ()
Vérifie et assure la connexion à la mémoire partagée.
- void `prise_deverouiller_trap` ()
Déverrouille la trappe et allume l'indication correspondante. permet d'avoir acces a la checkbox pour brancher la prise.
- void `prise_verouiller_trap` ()
Verrouille la trappe et éteint l'indication.
- void `prise_led_prise_on` ()
Allume la LED prise.
- void `prise_led_prise_off` ()
Éteint la LED prise.

Variables

- entrees * `io_p`
- int `shmid_p`

4.23.1 Description détaillée

Gestion de la prise et de la trappe de la borne.

Auteur

(Nshuti Nkurikiyinka Fidele, Mapako Eddy)

Version

1.0

Date

25 décembre 2025

Ce module permet de contrôler :

- l'état de la trappe (ouverte / fermée)
- la LED prise (prise connectée ou non)

Les accès aux E/S se font via la mémoire partagée, avec une initialisation automatique.

Définition dans le fichier `prise.c`.

4.23.2 Documentation des fonctions

4.23.2.1 prise_check_io()

```
int prise_check_io ()
```

Vérifie et assure la connexion à la mémoire partagée.

Renvoie

1 si io_p est valide, 0 sinon.

Définition à la ligne 47 du fichier [prise.c](#).

Voici le graphe d'appel pour cette fonction :



4.23.2.2 prise_deverouiller_trap()

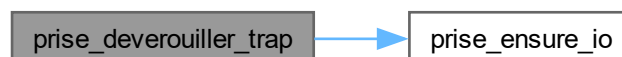
```
void prise_deverouiller_trap ()
```

Déverrouille la trappe et allume l'indication correspondante. permet d'avoir acces a la checkbox pour brancher la prise.

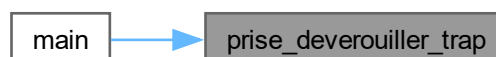
Utilisé lorsque l'utilisateur peut brancher ou retirer le câble.

Définition à la ligne 60 du fichier [prise.c](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



4.23.2.3 prise_ensure_io()

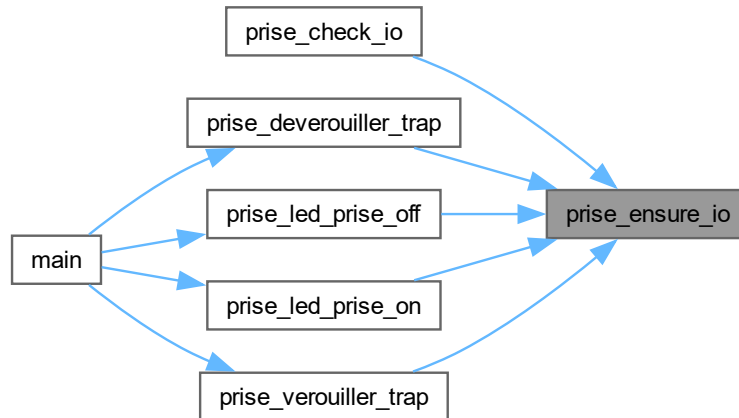
```
void prise_ensure_io ()
```

Initialise l'accès à la mémoire partagée si nécessaire.

Fonction de sécurité permettant d'éviter les accès NULL.

Définition à la ligne 33 du fichier [prise.c](#).

Voici le graphe des appelants de cette fonction :



4.23.2.4 `prise_led_praise_off()`

```
void prise_led_praise_off ()
```

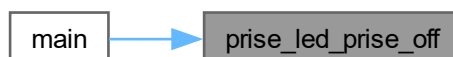
Éteint la LED prise.

Définition à la ligne 94 du fichier [prise.c](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



4.23.2.5 prise_led_prise_on()

```
void prise_led_prise_on ()
```

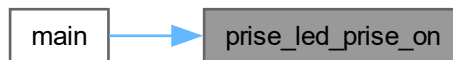
Allume la LED prise.

Définition à la ligne 83 du fichier [prise.c](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



4.23.2.6 prise_verouiller_trap()

```
void prise_verouiller_trap ()
```

Verrouille la trappe et éteint l'indication.

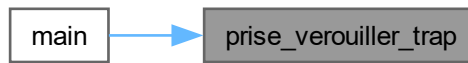
Utilisé lorsque le câble doit rester bloqué pendant la charge.

Définition à la ligne 73 du fichier [prise.c](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



4.23.3 Documentation des variables

4.23.3.1 io_p

entrees* io_p

Définition à la ligne 24 du fichier [prise.c](#).

4.23.3.2 shmid_p

int shmid_p

Définition à la ligne 25 du fichier [prise.c](#).

4.24 prise.c

[Aller à la documentation de ce fichier.](#)

```

00001
00015
00016
00017 #include <donnees_borne.h>
00018 #include <memoire_borne.h>
00019 #include "voyants.h"
00020 #include <stdio.h>
00021 #include "prise.h"
00022
00023 /* Pointeur vers la mémoire partagée */
00024 entrees* io_p;
00025 int shmid_p;
00026
00033 void prise_ensure_io()
00034 {
00035     if (io_p != NULL) return;
00036     io_p = acces_memoire(&shmid_p);
00037     if (io_p == NULL) {
00038         fprintf(stderr, "Erreur: acces memoire partagee impossible\n");
00039     }
00040 }
00041
00047 int prise_check_io()
00048 {
00049     if (io_p == NULL) prise_ensure_io();
00050     return (io_p != NULL);
00051 }
00052
00060 void prise_deverouiller_trap(){
00061
00062     if (io_p == NULL) prise_ensure_io();
00063     io_p->led_trappe = VERT;
00064 }
00065
00066
00073 void prise_verouiller_trap(){
00074
00075     if (io_p == NULL) prise_ensure_io();
00076     io_p->led_trappe = OFF;
00077 }
00078
00083 void prise_led_prise_on(){
00084
00085     if (io_p == NULL) prise_ensure_io();
  
```

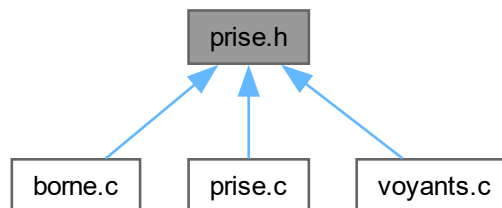
```

00086     io_p->led_prise = VERT;
00087
00088 }
00089
00094 void prise_led_prise_off() {
00095
00096     if (io_p == NULL) prise_ensure_io();
00097     io_p->led_prise = OFF;
00098
00099 }

```

4.25 Référence du fichier prise.h

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Fonctions

- int `prise_verrouiller_trappe` ()
- int `prise_check_io` ()
Vérifie et assure la connexion à la mémoire partagée.
- void `prise_ensure_io` ()
Initialise l'accès à la mémoire partagée si nécessaire.
- void `prise_deverrouiller_trap` ()
Déverrouille la trappe et allume l'indication correspondante. permet d'avoir acces a la checkbox pour brancher la prise.
- void `prise_verrouiller_trap` ()
Verrouille la trappe et éteint l'indication.
- void `prise_led_prise_on` ()
Allume la LED prise.
- void `prise_led_prise_off` ()
Éteint la LED prise.

4.25.1 Documentation des fonctions

4.25.1.1 `prise_check_io()`

```
int prise_check_io ()
```

Vérifie et assure la connexion à la mémoire partagée.

Renvoie

1 si io_p est valide, 0 sinon.

Définition à la ligne 47 du fichier [prise.c](#).

Voici le graphe d'appel pour cette fonction :

**4.25.1.2 prise_deverouiller_trap()**

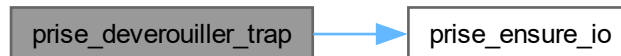
```
void prise_deverouiller_trap ()
```

Déverrouille la trappe et allume l'indication correspondante. permet d'avoir acces a la checkbox pour brancher la prise.

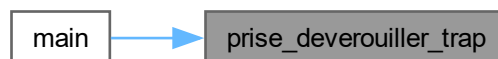
Utilisé lorsque l'utilisateur peut brancher ou retirer le câble.

Définition à la ligne 60 du fichier [prise.c](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

**4.25.1.3 prise_ensure_io()**

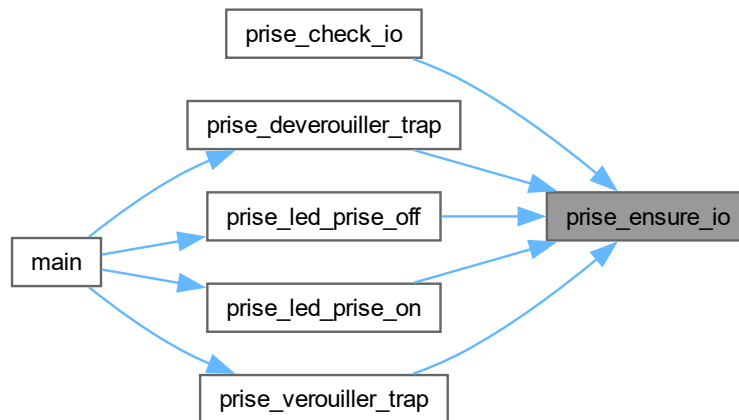
```
void prise_ensure_io ()
```

Initialise l'accès à la mémoire partagée si nécessaire.

Fonction de sécurité permettant d'éviter les accès NULL.

Définition à la ligne 33 du fichier [prise.c](#).

Voici le graphe des appelants de cette fonction :



4.25.1.4 `prise_led_prise_off()`

```
void prise_led_prise_off ()
```

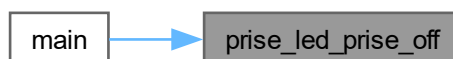
Éteint la LED prise.

Définition à la ligne 94 du fichier [prise.c](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



4.25.1.5 `prise_led_prise_on()`

```
void prise_led_prise_on ()
```

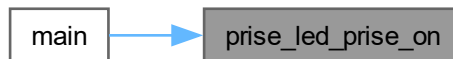
Allume la LED prise.

Définition à la ligne 83 du fichier [prise.c](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



4.25.1.6 prise_verouiller_trap()

```
void prise_verouiller_trap ()
```

Verrouille la trappe et éteint l'indication.

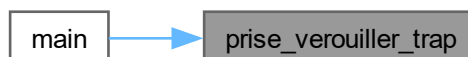
Utilisé lorsque le câble doit rester bloqué pendant la charge.

Définition à la ligne 73 du fichier [prise.c](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



4.25.1.7 prise_verouiller_trappe()

```
int prise_verouiller_trappe ()
```

4.26 prise.h

[Aller à la documentation de ce fichier.](#)

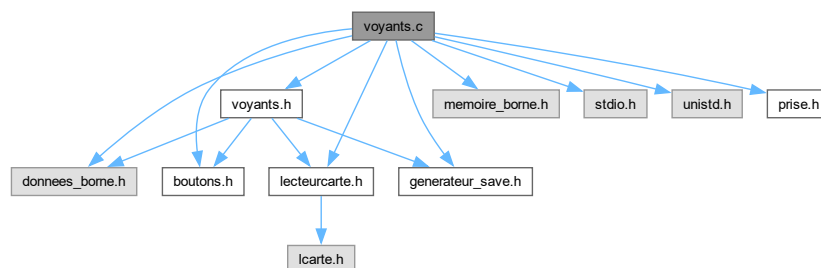
```
00001 #ifndef PRISE_H
00002 #define PRISE_H
00003 int prise_verouiller_trappe();
00004 int prise_check_io();
00005 void prise_ensure_io();
00006 void prise_deverouiller_trap();
00007 void prise_verouiller_trap();
00008 void prise_led_prise_on();
00009 void prise_led_prise_off();
00010 #endif
```

4.27 Référence du fichier voyants.c

Gestion des voyants de la borne de recharge.

```
#include <donnees_borne.h>
#include <memoire_borne.h>
#include "voyants.h"
#include <stdio.h>
#include "boutons.h"
#include <unistd.h>
#include "lecteurcarte.h"
#include "generateur_save.h"
#include "prise.h"
```

Graphe des dépendances par inclusion de voyants.c:



Fonctions

- void `voyants_ensure_io()`
Initialise l'accès à la mémoire partagée si nécessaire.
- void `voyants_set_charge_vert()`
Active la LED charge en VERT.
- void `voyants_set_charge_rouge()`
Active la LED charge en ROUGE.
- void `voyants_set_charge_off()`
Éteint la LED de charge.
- void `voyants_set_dispo_off()`
Éteint la LED dispo.
- void `voyants_set_dispo_on()`
Allume la LED dispo.
- void `voyants_init()`

- Initialise l'état par défaut des voyants.*
— void [voyants_toggle](#) ()
Fait clignoter la LED charge.
— void [voyants_toggle_default](#) ()
Fait clignoter la LED défaut en ROUGE.
— void [voyants_set_default_off](#) ()
Allumer la LED défaut.

Variables

- entrees * [io_l](#)
— int [shmid_l](#)

4.27.1 Description détaillée

Gestion des voyants de la borne de recharge.

Auteur

(Nshuti Nkurikiyinka Fidele, Mapako Eddy)

Version

1.0

Date

25 décembre 2025

Ce module permet de contrôler les voyants de la borne :

- LED CHARGE (OFF / VERT / ROUGE)
- LED DISPO (OFF / VERT)
- LED DEFAUT (OFF / ROUGE)
- LED PRISE
- LED TRAPPE

Fonctionnalités :

- Connexion automatique à la mémoire partagée
- Mise à jour des voyants
- Fonctions utilitaires d'initialisation et clignotement

Définition dans le fichier [voyants.c](#).

4.27.2 Documentation des fonctions

4.27.2.1 voyants_ensure_io()

```
void voyants_ensure_io ()
```

Initialise l'accès à la mémoire partagée si nécessaire.

Définition à la ligne 43 du fichier [voyants.c](#).

Voici le graphe des appelants de cette fonction :



4.27.2.2 voyants_init()

```
void voyants_init ()
```

Initialise l'état par défaut des voyants.

- Charge : OFF
- Dispo : VERT
- Défaut : OFF
- Prise : OFF
- Trappe : OFF

Définition à la ligne [122](#) du fichier [voyants.c](#).

Voici le graphe des appelants de cette fonction :



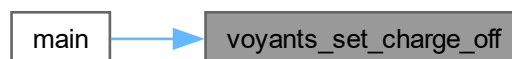
4.27.2.3 voyants_set_charge_off()

```
void voyants_set_charge_off ()
```

Éteint la LED de charge.

Définition à la ligne [88](#) du fichier [voyants.c](#).

Voici le graphe des appelants de cette fonction :



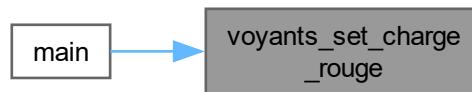
4.27.2.4 voyants_set_charge_rouge()

```
void voyants_set_charge_rouge ()
```

Active la LED charge en ROUGE .

Définition à la ligne [78](#) du fichier [voyants.c](#).

Voici le graphe des appelants de cette fonction :



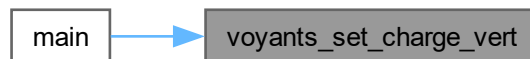
4.27.2.5 voyants_set_charge_vert()

```
void voyants_set_charge_vert ()
```

Active la LED charge en VERT.

Définition à la ligne 66 du fichier [voyants.c](#).

Voici le graphe des appelants de cette fonction :



4.27.2.6 voyants_set_default_off()

```
void voyants_set_default_off ()
```

Allumer la LED default.

Définition à la ligne 158 du fichier [voyants.c](#).

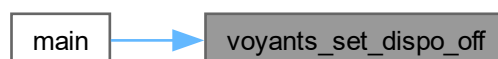
4.27.2.7 voyants_set_dispo_off()

```
void voyants_set_dispo_off ()
```

Éteint la LED dispo.

Définition à la ligne 98 du fichier [voyants.c](#).

Voici le graphe des appelants de cette fonction :



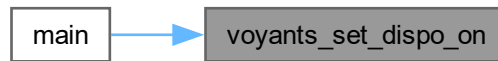
4.27.2.8 voyants_set_dispo_on()

```
void voyants_set_dispo_on ()
```

Allume la LED dispo.

Définition à la ligne 106 du fichier [voyants.c](#).

Voici le graphe des appelants de cette fonction :



4.27.2.9 voyants_toggle()

```
void voyants_toggle ()
```

Fait clignoter la LED charge.

Définition à la ligne 135 du fichier [voyants.c](#).

Voici le graphe des appelants de cette fonction :



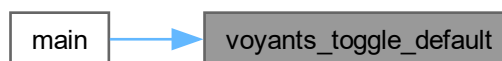
4.27.2.10 voyants_toggle_default()

```
void voyants_toggle_default ()
```

Fait clignoter la LED défaut en ROUGE.

Définition à la ligne 146 du fichier [voyants.c](#).

Voici le graphe des appelants de cette fonction :



4.27.3 Documentation des variables

4.27.3.1 io_l

```
entrees* io_l
```

Définition à la ligne 34 du fichier [voyants.c](#).

4.27.3.2 shmid_l

int shmid_l

Définition à la ligne 35 du fichier [voyants.c](#).

4.28 voyants.c

[Aller à la documentation de ce fichier.](#)

```

00001
00020
00021
00022
00023 #include <donnees_borne.h>
00024 #include <memoire_borne.h>
00025 #include "voyants.h"
00026 #include <stdio.h>
00027 #include "boutons.h"
00028 #include <unistd.h>
00029 #include "lecteurcarte.h"
00030 #include "generateur_save.h"
00031 #include "prise.h"
00032
00033 /* Pointeur vers la mémoire partagée */
00034 entrees* io_l;
00035 int shmid_l ;
00036
00037
00043 void voyants_ensure_io()
00044 {
00045     if (io_l != NULL) return;
00046     io_l = acces_memoire(&shmid_l);
00047     if (io_l == NULL) {
00048         fprintf(stderr, "Erreur: acces memoire partagee impossible\n");
00049     }
00050 }
00051
00057 static int voyants_check_io()
00058 {
00059     if (io_l == NULL) voyants_ensure_io();
00060     return (io_l != NULL);
00061 }
00066 void voyants_set_charge_vert() {
00067
00068     if (!voyants_check_io()) return; //erreur de segma
00069
00070     io_l->led_charge = VERT;
00071
00072 }
00073
00078 void voyants_set_charge_rouge() {
00079
00080     if (!voyants_check_io()) return;
00081     io_l->led_charge = ROUGE;
00082 }
00083
00088 void voyants_set_charge_off() {
00089
00090     if (!voyants_check_io()) return;
00091     io_l->led_prise = OFF;
00092 }
00093
00098 void voyants_set_dispo_off() {
00099     if (!voyants_check_io()) return;
00100     io_l->led_dispo = OFF;
00101 }
00106 void voyants_set_dispo_on() {
00107     if (!voyants_check_io()) return;
00108     io_l->led_dispo = VERT;
00109 }
00110
00111
00122 void voyants_init() {
00123     if (!voyants_check_io()) return;
00124     io_l->led_charge = OFF;
00125     io_l->led_dispo = VERT;
00126     io_l->led_default = OFF;
00127     io_l->led_prise = OFF;
00128     io_l->led_trappe = OFF;
00129 }

```

```

00130
00135 void voyants_toggle() {
00136     io_l->led_charge = VERT;
00137     sleep (1);
00138     io_l->led_charge = OFF;
00139     sleep (1);
00140 }
00141
00146 void voyants_toggle_default() {
00147     io_l->led_default = ROUGE;
00148     sleep (1);
00149     io_l->led_default = OFF;
00150     sleep (1);
00151 }
00152
00158 void voyants_set_default_off() {
00159     if (!voyants_check_io()) return;
00160     io_l->led_default = VERT;
00161 }

```

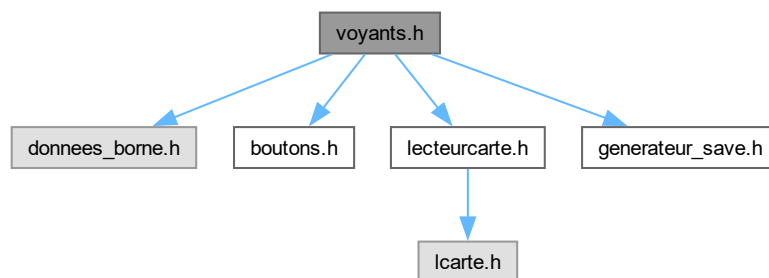
4.29 Référence du fichier voyants.h

```

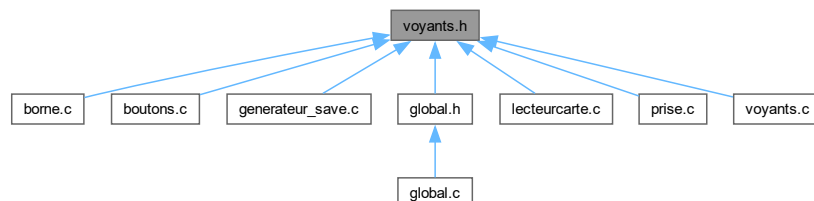
#include "donnees_borne.h"
#include "boutons.h"
#include "lecteurcarte.h"
#include "generateur_save.h"

```

Graphe des dépendances par inclusion de voyants.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Fonctions

- void `voyants_set_charge()`
- void `voyants_set_dispo_off()`
Éteint la LED dispo.
- void `voyants_set_dispo_on()`

Allume la LED dispo.
— void `voyants_toggle` ()
Fait clignoter la LED charge.
— void `voyants_ensure_io` ()
Initialise l'accès à la mémoire partagée si nécessaire.
— void `voyants_init` ()
Initialise l'état par défaut des voyants.
— void `voyants_set_charge_rouge` ()
Active la LED charge en ROUGE .
— void `voyants_set_charge_off` ()
Éteint la LED de charge.
— void `voyants_set_charge_vert` ()
Active la LED charge en VERT.
— void `voyants_toggle_default` ()
Fait clignoter la LED défaut en ROUGE.
— void `voyants_set_default_off` ()
Allumer la LED default.

4.29.1 Documentation des fonctions

4.29.1.1 `voyants_ensure_io()`

```
void voyants_ensure_io ()
```

Initialise l'accès à la mémoire partagée si nécessaire.

Définition à la ligne 43 du fichier `voyants.c`.

Voici le graphe des appelants de cette fonction :



4.29.1.2 `voyants_init()`

```
void voyants_init ()
```

Initialise l'état par défaut des voyants.

- Charge : OFF
- Dispo : VERT
- Défaut : OFF
- Prise : OFF
- Trappe : OFF

Définition à la ligne 122 du fichier `voyants.c`.

Voici le graphe des appelants de cette fonction :



4.29.1.3 voyants_set_charge()

```
void voyants_set_charge ()
```

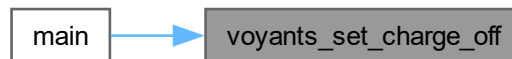
4.29.1.4 voyants_set_charge_off()

```
void voyants_set_charge_off ()
```

Éteint la LED de charge.

Définition à la ligne 88 du fichier [voyants.c](#).

Voici le graphe des appelants de cette fonction :



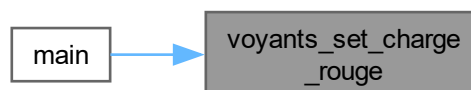
4.29.1.5 voyants_set_charge_rouge()

```
void voyants_set_charge_rouge ()
```

Active la LED charge en ROUGE .

Définition à la ligne 78 du fichier [voyants.c](#).

Voici le graphe des appelants de cette fonction :



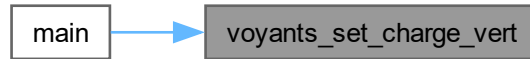
4.29.1.6 voyants_set_charge_vert()

```
void voyants_set_charge_vert ()
```

Active la LED charge en VERT.

Définition à la ligne 66 du fichier [voyants.c](#).

Voici le graphe des appelants de cette fonction :



4.29.1.7 voyants_set_default_off()

```
void voyants_set_default_off ()
```

Allumer la LED default.

Définition à la ligne 158 du fichier [voyants.c](#).

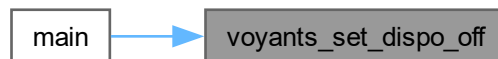
4.29.1.8 voyants_set_dispo_off()

```
void voyants_set_dispo_off ()
```

Éteint la LED dispo.

Définition à la ligne 98 du fichier [voyants.c](#).

Voici le graphe des appelants de cette fonction :



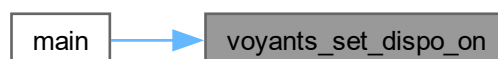
4.29.1.9 voyants_set_dispo_on()

```
void voyants_set_dispo_on ()
```

Allume la LED dispo.

Définition à la ligne 106 du fichier [voyants.c](#).

Voici le graphe des appelants de cette fonction :



4.29.1.10 voyants_toggle()

```
void voyants_toggle ()
```

Fait clignoter la LED charge.

Définition à la ligne 135 du fichier [voyants.c](#).

Voici le graphe des appelants de cette fonction :



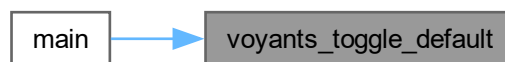
4.29.1.11 voyants_toggle_default()

```
void voyants_toggle_default ()
```

Fait clignoter la LED défaut en ROUGE.

Définition à la ligne 146 du fichier [voyants.c](#).

Voici le graphe des appelants de cette fonction :



4.30 voyants.h

[Aller à la documentation de ce fichier.](#)

```
00001 #ifndef VOYANTS_H
00002 #define VOYANTS_H
00003 #include "donnees_borne.h"
00004 #include "boutons.h"
00005 #include "lecteurcarte.h"
00006 #include "generateur_save.h"
00007 /* Contrôle des voyants (si mem == NULL utilise la memoire partagée interne) */
00008 void voyants_set_charge();
00009 void voyants_set_dispo_off();
00010 void voyants_set_dispo_on();
00011 void voyants_toggle();
00012 void voyants_ensure_io();
00013 void voyants_init();
00014 void voyants_set_charge_rouge();
00015 void voyants_set_charge_off();
00016 void voyants_set_charge_vert();
00017 void voyants_toggle_default();
00018 void voyants_set_default_off();
00019
00020 #endif
```