Projet de Gestion de Borne de Véhicule Électrique

Généré par Doxygen 1.9.1

Chapitre 1

Index des fichiers

1.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

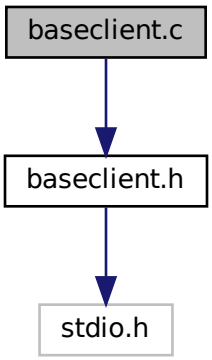
Chapitre 2

Documentation des fichiers

2.1 Référence du fichier baseclient.c

Ici on a les fonctions qui permettent de faire la gestion des clients.

`#include <baseclient.h>`
Graphe des dépendances par inclusion de baseclient.c:



Fonctions

- int baseclient_authentifier (int num)
  Authentifier le client qui se presente.
- void baseclient_ajoutclient (int k)
  Ajout un client dans la base de donnée.
- void baseclient_supprimeclient (int k)
  Supprime un client de la base de donnée.

2.1.1 Description détaillée

Ici on a les fonctions qui permettent de faire la gestion des clients.

2.1.2 Documentation des fonctions

2.1.2.1 baseclient_ajoutclient()

```
void baseclient_ajoutclient (
            int k )
```

Ajout un client dans la base de donnée.

Dans cette fonction on ouvre la base de données, on vérifie si le numéro client à ajouter n'est pas dans la base de donnée et après on l'ajoute. si le client est présent, on signal à l'opérateur que ce numéro est déjà présent dans la base de donnée.

Paramètres

in	*k*	c'est l'id du client à ajouter

Renvoie

void

variables internes utilisées

— fich : pointeur de type FILE vers le fichier de la base de donnée

2.1.2.2 baseclient_authentifier()

```
int baseclient_authentifier (
            int num )
```

Authentifier le client qui se presente.

Dans cette fonction on ouvre la base de données database.txt on vérifie si le numéro entrer par le client est présent dans la base de donnée.

Paramètres

in	*num*	c'est l'id du client à faire valider

Renvoie

int

Valeurs retournées

1	si le client est dans la base de données
0	si le client ne figure pas dans la base de donné

2.1.2.3 baseclient_supprimeclient()

```
void baseclient_supprimeclient (
            int k )
```

Supprime un client de la base de donnée.

Dans cette fonction on créer un fichier temporaire et à la fin on supprime l'ancienne base de donnée et on renome le fichier temporaire en notre fichier base de donnée. Ce fichier temporaire nous permet de transférer les numeros de clients qui sont différent du numéro à supprimer. Si le client à supprimer ne figure pas dans la base de donnée on le notifie à l'opérateur Si la suppression a reussit, on le signale à l'opérateur

Paramètres

in	*k*	c'est l'id du client à supprimer

Renvoie

void

variables internes utilisées

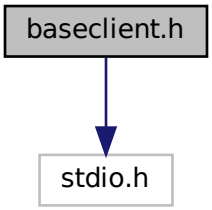
- fich : pointeur de type FILE vers le fichier de la base de donnée
- tmp_data : pointeur de type FILE vers le fichier temporaire
- val : variable qui permet de lire les numéros dans la base de donnée
- found : variable qui permet de savoir si le client à supprimer est dans la base de donnée ou pas

2.2 Référence du fichier baseclient.h

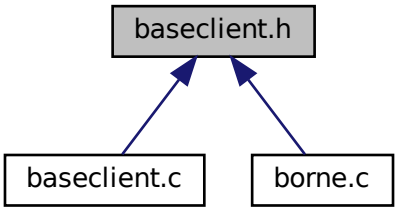
Dans ce fichier on déclare les fonctions pour la gestion des clients.

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

Graphe des dépendances par inclusion de baseclient.h:



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



Fonctions

- int baseclient_authentifier (int)
  Authentifier le client qui se presente.
- void baseclient_ajoutclient (int)
  Ajout un client dans la base de donnée.
- void baseclient_supprimeclient (int)
  Supprime un client de la base de donnée.

2.2.1 Description détaillée

Dans ce fichier on déclare les fonctions pour la gestion des clients.

2.2.2 Documentation des fonctions

2.2.2.1 baseclient_ajoutclient()

```
void baseclient_ajoutclient (
            int k )
```

Ajout un client dans la base de donnée.

Dans cette fonction on ouvre la base de données, on vérifie si le numéro client à ajouter n'est pas dans la base de donnée et après on l'ajoute. si le client est présent, on signal à l'opérateur que ce numéro est déjà présent dans la base de donnée.

Paramètres

in	*k*	c'est l'id du client à ajouter

Renvoie

void

variables internes utilisées

— fich : pointeur de type FILE vers le fichier de la base de donnée

2.2.2.2 baseclient_authentifier()

```
int baseclient_authentifier (
            int num )
```

Authentifier le client qui se presente.

Dans cette fonction on ouvre la base de données database.txt on vérifie si le numéro entrer par le client est présent dans la base de donnée.

Paramètres

in	*num*	c'est l'id du client à faire valider

Renvoie

int

Valeurs retournées

1	si le client est dans la base de données
0	si le client ne figure pas dans la base de donné

2.2.2.3 baseclient_supprimeclient()

```
void baseclient_supprimeclient (
            int k )
```

Supprime un client de la base de donnée.

Dans cette fonction on créer un fichier temporaire et à la fin on supprime l'ancienne base de donnée et on renome le fichier temporaire en notre fichier base de donnée. Ce fichier temporaire nous permet de transférer les numeros de clients qui sont différent du numéro à supprimer. Si le client à supprimer ne figure pas dans la base de donnée on le notifie à l'opérateur Si la suppression a reussit, on le signale à l'opérateur

Paramètres

in	*k*	c'est l'id du client à supprimer

Renvoie

void

variables internes utilisées

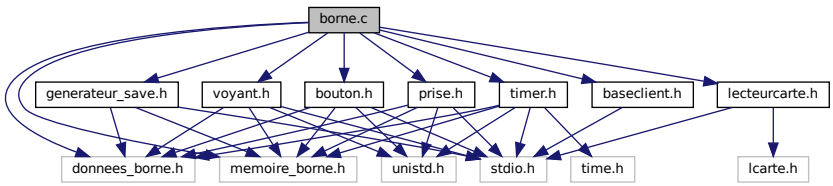
- fich : pointeur de type FILE vers le fichier de la base de donnée
- tmp_data : pointeur de type FILE vers le fichier temporaire
- val : variable qui permet de lire les numéros dans la base de donnée
- found : variable qui permet de savoir si le client à supprimer est dans la base de donnée ou pas

2.3 Référence du fichier borne.c

borne.c est le fichier principale du système

```
#include <memoire_borne.h>
#include <donnees_borne.h>
#include "lecteurcarte.h"
#include "baseclient.h"
#include "voyant.h"
#include "timer.h"
#include "bouton.h"
#include "prise.h"
#include "generateur_save.h"
```

Graphe des dépendances par inclusion de borne.c:



Énumérations

- enum etatsystem {
  etat0 , etat1 , etat2 , etat3 ,
  etat4 , etat5 , etat6 , etat255 }
  Définition des états du système de recharge On déclare une énumération pour représenter les différents états du système de recharge de véhicule électrique.

Fonctions

- void administration_operateur ()
  Fonction charger de gérer l'administration des clients une fois que l'operateur met son id.
- int main ()
  la fonction principale de tout le système, c'est ici qu'est implementée la machine à état decrivant le système

2.3.1 Description détaillée

borne.c est le fichier principale du système

Auteur

KAMDA TEZEBO DIBREY JONATAN
Nassime BOUANANI

Version

1.0

Date

03/12/2025

2.3.2 Documentation du type de l'énumération

2.3.2.1 etatsystem

`enum etatsystem`

Définition des états du système de recharge On déclare une énumération pour représenter les différents états du système de recharge de véhicule électrique.

Valeurs énumérées

etat0	
etat1	
etat2	
etat3	
etat4	
etat5	
etat6	
etat255	

2.3.3 Documentation des fonctions

2.3.3.1 administration_operateur()

`void administration_operateur ( )`

Fonction charger de gérer l'administration des clients une fois que l'operateur met son id.

Note

Cette fonction est utilisé pour le cas etat255

Cette fonction demande à l'opérateur de saisir le choix de l'opération qu'il aimerait faire soit ajout ou suppression de client par la suite, l'opérateur fait entrer le numéro du client correspondant ensuite elle fait appels aux fonctions de baseclient_ajoutclient(int) ou baseclient_supprimeclient(int) selon le besoin.

Variables internes utilisées :

— c : variable de type int pour stocker le choix de l'opération et le numéro du client

Renvoie

void

2.3.3.2 main()

```
int main ( )
```

la fonction principale de tout le système, c'est ici qu'est implementée la machine à état decrivant le système

Note

dans le main, une boucle infinie est faite !

Dans cette fonction on implemente la machine à état en utilisant switch case et etatsystem. Les cas sont extraites de la machine à état. dans la fonction on definit deux états, état_present et état_suivant Pour évoluer d'un état à un autre, état_present prend la valeur de état_suivant état0 on demande à l'utilisateur d'entrer son numéro et s'assure qu'il clique sur button charge en moins de 1 min état1 on attent que l'utilisateur branche la prise état2 c'est la prise branché état3 c'est la charge du véhicule état4 c'est la fin de charge état5 c'est la récupération du véhicule état6 c'est la mise en place du système comme s'il était à l'état initiale état255 c'est l'état de gestion de l'administration par l'opérateur

Avertissement

l'état initiale dont fait référence l'état6 est différent de état0 car c'est l'état de démarrage du système avant l'attente des identifiants du client.

Variables internes utilisées :

— etat_present : variable de type etatsystem qui stocke l'état actuel du système
— etat_suivant : variable de type etatsystem qui stocke l'état suivant du système
— id : variable de type int utilisée pour la gestion de l'authentification lors de la reprise du véhicule
— data : variable entière int pour stocker le numéro entré par l'utilisateur lors de l'authentification
— numero : variable entière int pour stocker le numéro lu par le lecteur de carte
— found : variable entière int pour stocker le résultat de l'authentification du client
— timer_secs : variable entière int pour stocker le temps écoulé en secondes
— butt_apuie : variable entière int pour vérifier si le bouton charge a été appuyé

Renvoie

int

Valeurs retournées

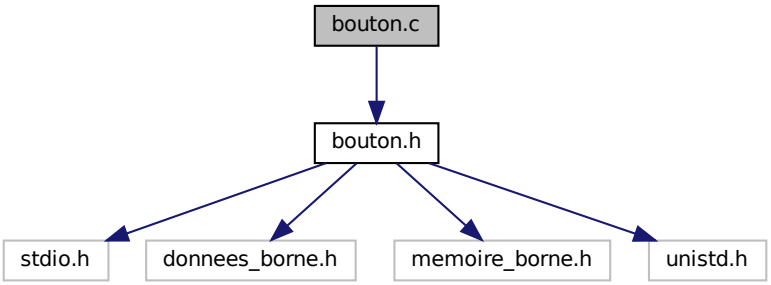
0	en cas de succès d'execution

2.4 Référence du fichier bouton.c

Ici on a les fonctions qui permettent de lire la valeur du bouton et de mettre les états à 0.

```
#include "bouton.h"
```

Graphe des dépendances par inclusion de bouton.c:



Fonctions

- int bouton_appuie_button_charge ()
  Cette fonction retourne la valeur du bouton charge.
- int bouton_apppuie_button_stop ()
  Cette fonction retourne la valeur du bouton stop.
- void bouton_set_bouton_charge ()
  Cette fonction met l'état du bouton charge à 0.
- void bouton_set_bouton_stop ()
  Cette fonction met l'état du bouton stop à 0.

Variables

- entrees * io_b
  variable globale de bouton.c pour la gestion mémoire partagée
- int shmid_b
  variable globale de bouton.c

2.4.1 Description détaillée

Ici on a les fonctions qui permettent de lire la valeur du bouton et de mettre les états à 0.

2.4.2 Documentation des fonctions

2.4.2.1 bouton_apppuie_button_stop()

```
int bouton_apppuie_button_stop ( )
```

Cette fonction retourne la valeur du bouton stop.

On accède la mémoire partagée shmid et ensuite on lit dans cette mémoire la valeur du bouton stop

Renvoie

int

Valeurs retournées

1	si le client a appuyer
0	si le client n'a pas appuyer

2.4.2.2 bouton_appuie_button_charge()

```
int bouton_appuie_button_charge ( )
```

Cette fonction retourne la valeur du bouton charge.

On accède la mémoire partagée shmid et ensuite on lit dans cette mémoire la valeur de la charge

Renvoie

int

Valeurs retournées

1	si le client a appuyer
0	si le client n'a pas appuyer

2.4.2.3 bouton_set_bouton_charge()

```
void bouton_set_bouton_charge ( )
```

Cette fonction met l'état du bouton charge à 0.

On accède la mémoire partagée shmid et on écrit 0 sur cette mémoire

Renvoie

void

2.4.2.4 bouton_set_bouton_stop()

```
void bouton_set_bouton_stop ( )
```

Cette fonction met l'état du bouton stop à 0.

On accède la mémoire partagée shmid et on écrit 0 sur cette mémoire

Renvoie

void

2.4.3 Documentation des variables

2.4.3.1 io_b

```
entrees* io_b
```

variable globale de bouton.c pour la gestion mémoire partagée

- *io_b: pointeur vers la structure des entrées de la mémoire partagée

2.4.3.2 shmid_b

```
int shmid_b
```

variable globale de bouton.c

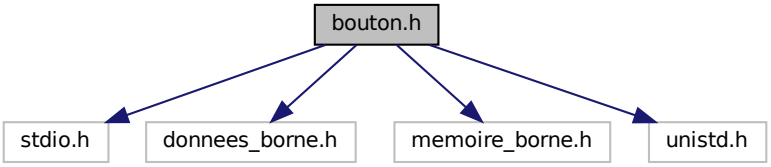
- *shmid_b : entier qui servira d'entrée pour la mémoire partagée

2.5 Référence du fichier bouton.h

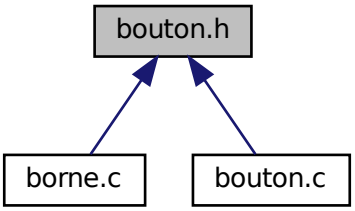
Dans ce fichier on déclare les fonctions pour la gestion des boutons.

```
#include <stdio.h>
#include <donnees_borne.h>
#include <memoire_borne.h>
#include <unistd.h>
```

Graphe des dépendances par inclusion de bouton.h:



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



Fonctions

- int bouton_appuie_button_charge ()
  Cette fonction retourne la valeur du bouton charge.
- int bouton_apppuie_button_stop ()
  Cette fonction retourne la valeur du bouton stop.
- void bouton_set_bouton_charge ()
  Cette fonction met l'état du bouton charge à 0.
- void bouton_set_bouton_stop ()
  Cette fonction met l'état du bouton stop à 0.

2.5.1 Description détaillée

Dans ce fichier on déclare les fonctions pour la gestion des boutons.

2.5.2 Documentation des fonctions

2.5.2.1 bouton_apppuie_button_stop()

```
int bouton_apppuie_button_stop ( )
```

Cette fonction retourne la valeur du bouton stop.

On accède la mémoire partagée shmid et ensuite on lit dans cette mémoire la valeur du bouton stop

Renvoie

int

Valeurs retournées

1	si le client a appuyer
0	si le client n'a pas appuyer

2.5.2.2 bouton_appuie_button_charge()

```
int bouton_appuie_button_charge ( )
```

Cette fonction retourne la valeur du bouton charge.

On accède la mémoire partagée shmid et ensuite on lit dans cette mémoire la valeur de la charge

Renvoie

int

Valeurs retournées

1	si le client a appuyer
0	si le client n'a pas appuyer

2.5.2.3 bouton_set_bouton_charge()

```
void bouton_set_bouton_charge ( )
```

Cette fonction met l'état du bouton charge à 0.

On accède la mémoire partagée shmid et on écrit 0 sur cette mémoire

Renvoie

void

2.5.2.4 bouton_set_bouton_stop()

```
void bouton_set_bouton_stop ( )
```

Cette fonction met l'état du bouton stop à 0.

On accède la mémoire partagée shmid et on écrit 0 sur cette mémoire

Renvoie

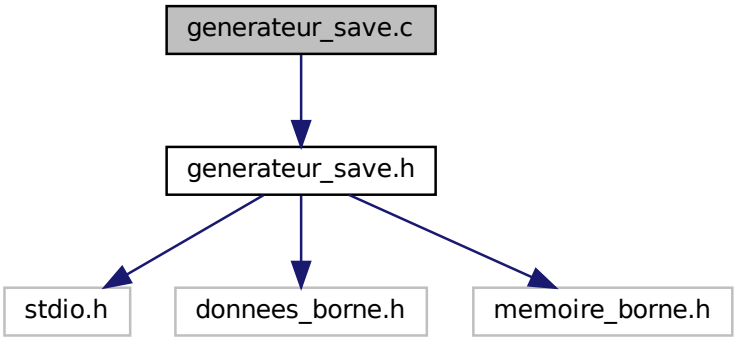
void

2.6 Référence du fichier generateur_save.c

Dans ce fichier on fait la gestion du contacteur et du generateur save.

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

Graphe des dépendances par inclusion de generateur_save.c:



Fonctions

- void generateur_save_fermer_contacteur ()
  Cette fonction ferme le contacteur.
- void generateur_save_ouvrir_contacteur ()
  Cette fonction ouvre le contacteur.
- int generateur_save_tension_DC ()
  Cette fonction consulte la valeur de la tension DC et la renvoie.
- void generateur_save_generer_pwm (pwm signal)
  Cette fonction génère le signal PWM.

Variables

— entrees * io_g
variable globale de generateur_save.c pour la gestion mémoire partagée
— int shmid_g
variable globale de generateur_save.c

2.6.1 Description détaillée

Dans ce fichier on fait la gestion du contacteur et du generateur save.

2.6.2 Documentation des fonctions

2.6.2.1 generateur_save_fermer_contacteur()

```
void generateur_save_fermer_contacteur ( )
```

Cette fonction ferme le contacteur.

accède la mémoire partagée partagé shmid et on ferme le contacteur en mettant la valeur à 1

Renvoie

void

2.6.2.2 generateur_save_generer_pwm()

```
void generateur_save_generer_pwm (
            pwm signal )
```

Cette fonction génère le signal PWM.

accède la mémoire partagée partagé shmid et ensuite on écrit dans cette mémoire.

Renvoie

void

2.6.2.3 generateur_save_ouvrir_contacteur()

```
void generateur_save_ouvrir_contacteur ( )
```

Cette fonction ouvre le contacteur.

accède la mémoire partagée partagé shmid et ensuite on écrit dans cette mémoire pour ouvrir le contacteur en mettant la valeur à 0

Renvoie

void

2.6.2.4 generateur_save_tension_DC()

```
int generateur_save_tension_DC ( )
```

Cette fonction consulte la valeur de la tension DC et la renvoie.

accède la mémoire partagée partagé shmid et ensuite on lit dans cette mémoire et on retourne la valeur lu. les valeurs de tension possible générer par le générateur save est 12,9,6,0.

Renvoie

int

Valeurs retournées

12	
9	
6	
0	

2.6.3 Documentation des variables

2.6.3.1 io_g

```
entrees* io_g
```

variable globale de generateur_save.c pour la gestion mémoire partagée

— ∗io_g: pointeur vers la structure des entrées de la mémoire partagée

2.6.3.2 shmid_g

```
int shmid_g
```

variable globale de generateur_save.c

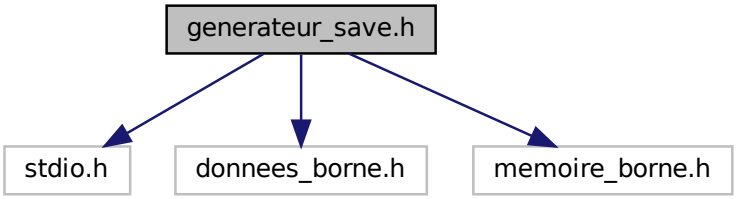
— ∗shmid_g : entier qui servira d'entrée pour la mémoire partagée

2.7 Référence du fichier generateur_save.h

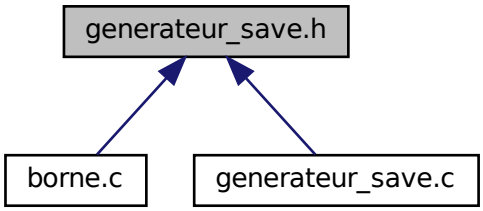
Dans ce fichier on déclare les fonctions pour la gestion du générateur save.

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

Graphe des dépendances par inclusion de generateur_save.h:



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



Fonctions

— void generateur_save_fermer_contacteur ()
Cette fonction ferme le contacteur.
— void generateur_save_ouvrir_contacteur ()
Cette fonction ouvre le contacteur.
— int generateur_save_tension_DC ()
Cette fonction consulte la valeur de la tension DC et la renvoie.
— void generateur_save_generer_pwm (pwm signal)
Cette fonction génère le signal PWM.

2.7.1 Description détaillée

Dans ce fichier on déclare les fonctions pour la gestion du générateur save.

2.7.2 Documentation des fonctions

2.7.2.1 generateur_save_fermer_contacteur()

```
void generateur_save_fermer_contacteur ( )
```

Cette fonction ferme le contacteur.

accède la mémoire partagée partagé shmid et on ferme le contacteur en mettant la valeur à 1

Renvoie

void

2.7.2.2 generateur_save_generer_pwm()

```
void generateur_save_generer_pwm (
            pwm signal )
```

Cette fonction génère le signal PWM.

accède la mémoire partagée partagé shmid et ensuite on écrit dans cette mémoire.

Renvoie

void

2.7.2.3 generateur_save_ouvrir_contacteur()

```
void generateur_save_ouvrir_contacteur ( )
```

Cette fonction ouvre le contacteur.

accède la mémoire partagée partagé shmid et ensuite on écrit dans cette mémoire pour ouvrir le contacteur en mettant la valeur à 0

Renvoie

void

2.7.2.4 generateur_save_tension_DC()

```
int generateur_save_tension_DC ( )
```

Cette fonction consulte la valeur de la tension DC et la renvoie.

accède la mémoire partagée partagé shmid et ensuite on lit dans cette mémoire et on retourne la valeur lu. les valeurs de tension possible générer par le générateur save est 12,9,6,0.

Renvoie

int

Valeurs retournées

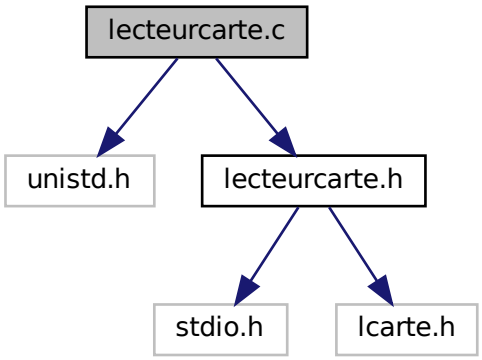
12	
9	
6	
0	

2.8 Référence du fichier lecteurcarte.c

Dans ce fichier on fait la gestion du lecteur de carte.

```
#include <unistd.h>
#include "lecteurcarte.h"
```

Graphe des dépendances par inclusion de lecteurcarte.c:



Fonctions

- void lecteurcarte_initialiser_lecteur ()
  Cette fonction permet d'initialiser le port sur le lecteur.
- void lecteurcarte_initialiser ()
  Cette fonction initialise le lecteur de carte.
- int lecteurcarte_lire_carte ()
  Cette fonction lit la valeur du numéro passé en paramètre.

2.8.1 Description détaillée

Dans ce fichier on fait la gestion du lecteur de carte.

2.8.2 Documentation des fonctions

2.8.2.1 lecteurcarte_initialiser()

```
void lecteurcarte_initialiser ( )
```

Cette fonction initialise le lecteur de carte.

Cette fonction affiche un message demandant a l'utilisateur d'insérer une carte

Renvoie

void

2.8.2.2 lecteurcarte_initialiser_lecteur()

```
void lecteurcarte_initialiser_lecteur ( )
```

Cette fonction permet d'initialiser le port sur le lecteur.

Cette fonction est déjà écrite par le prof elle est juste exploitée dans notre code

Renvoie

void

2.8.2.3 lecteurcarte_lire_carte()

```
int lecteurcarte_lire_carte ( )
```

Cette fonction lit la valeur du numéro passé en paramètre.

Cette fonction affiche un message demandant a l'utilisateur d'insérer une carte

variable internes utilisées

int num : le numero de la carte insérée

Renvoie

int le numero de la carte inserée

Valeurs retournées

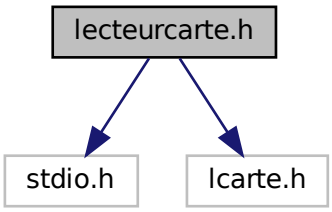
num	le numero de la carte inserée

2.9 Référence du fichier lecteurcarte.h

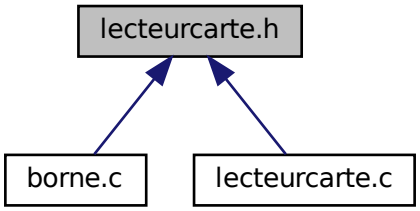
Dans ce fichier on déclare les fonctions pour la gestion du lecteur de carte.

```
#include <stdio.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_lecteur ()
  Cette fonction permet d'initialiser le port sur le lecteur.
- void lecteurcarte_initialiser ()
  Cette fonction initialise le lecteur de carte.
- int lecteurcarte_lire_carte ()
  Cette fonction lit la valeur du numéro passé en paramètre.

2.9.1 Description détaillée

Dans ce fichier on déclare les fonctions pour la gestion du lecteur de carte.

2.9.2 Documentation des fonctions

2.9.2.1 lecteurcarte_initialiser()

```
void lecteurcarte_initialiser ( )
```

Cette fonction initialise le lecteur de carte.

Cette fonction affiche un message demandant a l'utilisateur d'insérer une carte

Renvoie

void

2.9.2.2 lecteurcarte_initialiser_lecteur()

```
void lecteurcarte_initialiser_lecteur ( )
```

Cette fonction permet d'initialiser le port sur le lecteur.

Cette fonction est déjà écrite par le prof elle est juste exploitée dans notre code

Renvoie

void

2.9.2.3 lecteurcarte_lire_carte()

```
int lecteurcarte_lire_carte ( )
```

Cette fonction lit la valeur du numéro passé en paramètre.

Cette fonction affiche un message demandant a l'utilisateur d'insérer une carte

variable internes utilisées

int num : le numero de la carte insérée

Renvoie

int le numero de la carte inserée

Valeurs retournées

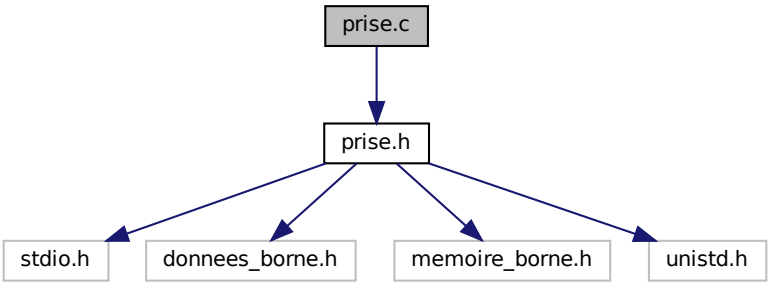
num	le numero de la carte inserée

2.10 Référence du fichier prise.c

Dans ce fichier on fait la gestion de la prise.

```
#include "prise.h"
```

Graphe des dépendances par inclusion de prise.c:



Fonctions

- void prise_verrouille_trappe ()
  Cette fonction verouille la trappe.
- void prise_deverrouille_trappe ()
  Cette fonction verouille la trappe.
- void prise_set_prise (led prise)
  Cette fonction met la led à VERT ou ROUGE ou l'éteint.

Variables

- entrees * io_p
  variable globale de prise.c pour la gestion mémoire partagée
- int shmid_p
  variable globale de prise.c

2.10.1 Description détaillée

Dans ce fichier on fait la gestion de la prise.

2.10.2 Documentation des fonctions

2.10.2.1 prise_deverrouille_trappe()

```
void prise_deverrouille_trappe ( )
```

Cette fonction verouille la trappe.

Dans cette fonction on accède la mémoire partagée et on dévérouille la trappe en mettant la led à VERT

Renvoie

void

2.10.2.2 prise_set_prise()

```
void prise_set_prise (
            led prise )
```

Cette fonction met la led à VERT ou ROUGE ou l'éteint.

Dans cette fonction on accède la mémoire partagée et on met la led à l'état souhaité

Paramètres

in	*prise*	l'état souhaité de la led

Renvoie

void

2.10.2.3 prise_verrouille_trappe()

```
void prise_verrouille_trappe ( )
```

Cette fonction verouille la trappe.

Dans cette fonction on accède la mémoire partagée et on vérouille la trappe en mettant la led à OFF

Renvoie

void

2.10.3 Documentation des variables

2.10.3.1 io_p

`entrees* io_p`

variable globale de prise.c pour la gestion mémoire partagée

— *io_p: pointeur vers la structure des entrées de la mémoire partagée

2.10.3.2 shmid_p

`int shmid_p`

variable globale de prise.c

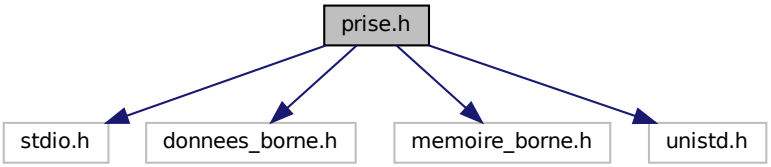
— *shmid_p : entier qui servira d'entrée pour la mémoire partagée

2.11 Référence du fichier prise.h

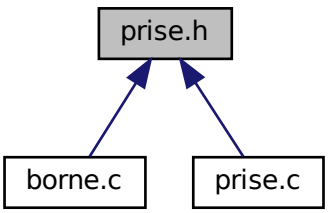
Dans ce fichier on déclare les fonctions pour la gestion de la prise.

```
#include <stdio.h>
#include <donnees_borne.h>
#include <memoire_borne.h>
#include <unistd.h>
```

Graphe des dépendances par inclusion de prise.h:



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



Fonctions

- void prise_verrouille_trappe ()
  Cette fonction verouille la trappe.
- void prise_deverrouille_trappe ()
  Cette fonction verouille la trappe.
- void prise_set_prise (led prise)
  Cette fonction met la led à VERT ou ROUGE ou l'éteint.

2.11.1 Description détaillée

Dans ce fichier on déclare les fonctions pour la gestion de la prise.

2.11.2 Documentation des fonctions

2.11.2.1 prise_deverrouille_trappe()

```
void prise_deverrouille_trappe ( )
```

Cette fonction verouille la trappe.

Dans cette fonction on accède la mémoire partagée et on dévérouille la trappe en mettant la led à VERT

Renvoie

void

2.11.2.2 prise_set_prise()

```
void prise_set_prise (
            led prise )
```

Cette fonction met la led à VERT ou ROUGE ou l'éteint.

Dans cette fonction on accède la mémoire partagée et on met la led à l'état souhaité

Paramètres

in	*prise*	l'état souhaité de la led

Renvoie

void

2.11.2.3 prise_verrouille_trappe()

```
void prise_verrouille_trappe ( )
```

Cette fonction verouille la trappe.

Dans cette fonction on accède la mémoire partagée et on vérouille la trappe en mettant la led à OFF

Renvoie

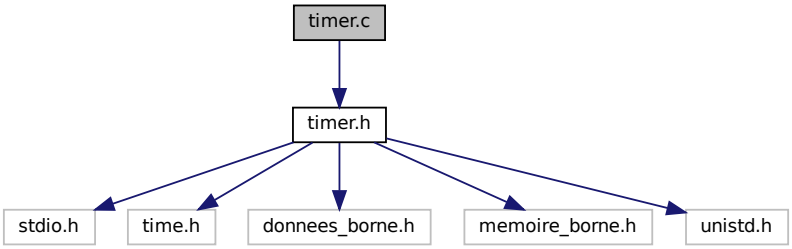
void

2.12 Référence du fichier timer.c

Dans ce fichier on fait la gestion du timer.

```
#include "timer.h"
```

Graphe des dépendances par inclusion de timer.c:



Fonctions

- void timer_raz ()
  Cette fonction remet le timer à zéro
- int timer_valeur ()
  Cette fonction calcule la différence de temps depuis le dernier appel
- int timer_count_sec ()
  Cette fonction met à jour le timer chaque seconde.

Variables

- entrees * io_t
  variable globale de timer.c
- int shmid_t
  variable globale de timer.c
- time_t depart_timer
  variable globale de timer.c

2.12.1 Description détaillée

Dans ce fichier on fait la gestion du timer.

2.12.2 Documentation des fonctions

2.12.2.1 timer_count_sec()

```
int timer_count_sec ( )
```

Cette fonction met à jour le timer chaque seconde.

Dans cette fonction on accède la mémoire partagée et on met à jour le timer La valeur du timer est incrémentée chaque seconde La fonction timer_valeur() est appelée pour obtenir la seconde écoulée depuis le dernier appel.

Variable interne utilisée :

— timer_value : variable de type int pour stocker la valeur du timer en secondes

Renvoie

int

Valeurs retournées

int	le nombre de secondes écoulées

2.12.2.2 timer_raz()

```
void timer_raz ( )
```

Cette fonction remet le timer à zéro

Dans cette fonction on accède la mémoire partagée et on remet le timer à zéro La variable depart_timer est initialisée avec le temps actuel

Renvoie

void

2.12.2.3 timer_valeur()

```
int timer_valeur ( )
```

Cette fonction calcule la différence de temps depuis le dernier appel

Dans cette fonction on dort pendant 1 seconde puis on calcule la différence entre le temps actuel et le temps de départ La variable depart_timer est mise à jour avec le temps actuel pour le prochain appel.

Variable interne utilisée :

- now : variable de type time_t pour stocker le temps actuel
- diff : variable de type int pour stocker la différence de temps en secondes

Renvoie

int

Valeurs retournées

int	la différence de temps en secondes

2.12.3 Documentation des variables

2.12.3.1 depart_timer

```
time_t depart_timer
```

variable globale de timer.c

- depart_timer : variable de type time_t pour stocker le temps de départ du timer

2.12.3.2 io_t

```
entrees* io_t
```

variable globale de timer.c

- ∗io_t: pointeur vers la structure des entrées de la mémoire partagée

2.12.3.3 shmid_t

```
int shmid_t
```

variable globale de timer.c

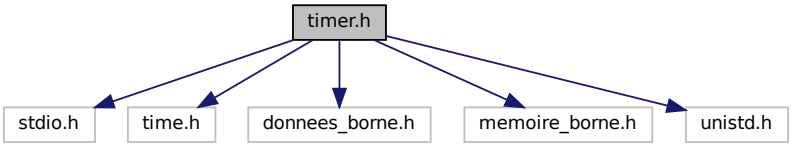
— ∗shmid_t : entier qui servira d'entrée pour la mémoire partagée

2.13 Référence du fichier timer.h

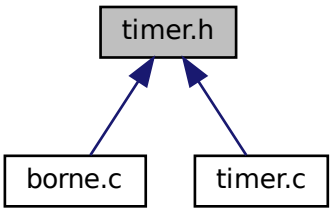
Dans ce fichier on déclare les fonctions pour la gestion du timer.

```
#include <stdio.h>
#include <time.h>
#include <donnees_borne.h>
#include <memoire_borne.h>
#include <unistd.h>
```

Graphe des dépendances par inclusion de timer.h:



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



Fonctions

— void timer_raz ()
Cette fonction remet le timer à zéro

— int timer_valeur ()
Cette fonction calcule la différence de temps depuis le dernier appel

— int timer_count_sec ()
Cette fonction met à jour le timer chaque seconde.

2.13.1 Description détaillée

Dans ce fichier on déclare les fonctions pour la gestion du timer.

2.13.2 Documentation des fonctions

2.13.2.1 timer_count_sec()

```
int timer_count_sec ( )
```

Cette fonction met à jour le timer chaque seconde.

Dans cette fonction on accède la mémoire partagée et on met à jour le timer La valeur du timer est incrémentée chaque seconde La fonction timer_valeur() est appelée pour obtenir la seconde écoulée depuis le dernier appel.

Variable interne utilisée :

— timer_value : variable de type int pour stocker la valeur du timer en secondes

Renvoie

int

Valeurs retournées

int	le nombre de secondes écoulées

2.13.2.2 timer_raz()

```
void timer_raz ( )
```

Cette fonction remet le timer à zéro

Dans cette fonction on accède la mémoire partagée et on remet le timer à zéro La variable depart_timer est initialisée avec le temps actuel

Renvoie

void

2.13.2.3 timer_valeur()

```
int timer_valeur ( )
```

Cette fonction calcule la différence de temps depuis le dernier appel

Dans cette fonction on dort pendant 1 seconde puis on calcule la différence entre le temps actuel et le temps de départ La variable depart_timer est mise à jour avec le temps actuel pour le prochain appel.

Variable interne utilisée :

- now : variable de type time_t pour stocker le temps actuel
- diff : variable de type int pour stocker la différence de temps en secondes

Renvoie

int

Valeurs retournées

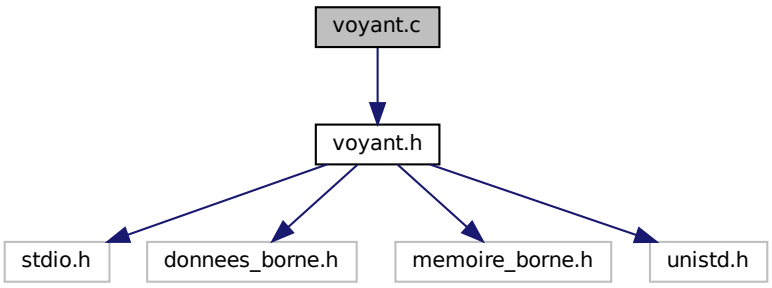
int	la différence de temps en secondes

2.14 Référence du fichier voyant.c

Dans ce fichier on fait la gestion des voyants.

```
#include "voyant.h"
```

Graphe des dépendances par inclusion de voyant.c:



Fonctions

- void voyant_set_dispo (led dispo)
  Cette fonction met la led de disponibilité à VERT ou ROUGE ou l'éteint.
- void voyant_set_charge (led charge)

Cette fonction met la led de charge à VERT ou ROUGE ou l'éteint.

— void voyant_blink_charge ()
Cette fonction fait clignoter la led de charge.
— void voyant_set_defaut (led defaut)
Cette fonction met la led de défaut à VERT ou ROUGE ou l'éteint.
— void voyant_blink_defaut ()
Cette fonction fait clignoter la led de défaut.
— void voyant_set_prise (led prise)
Cette fonction met la led de prise à VERT ou ROUGE ou l'éteint.
— void voyant_set_trappe (led trappe)
Cette fonction met la led de trappe à VERT ou ROUGE ou l'éteint.

Variables

— entrees * io
variable globale de voyant.c pour la gestion mémoire partagée
— int shmid
variable globale de voyant.c

2.14.1 Description détaillée

Dans ce fichier on fait la gestion des voyants.

2.14.2 Documentation des fonctions

2.14.2.1 voyant_blink_charge()

```
void voyant_blink_charge ( )
```

Cette fonction fait clignoter la led de charge.

Dans cette fonction on fait clignoter la led de charge en VERT 4 fois avec un intervalle d'une seconde Ce qui donne 8 secondes de clignotement. 1 seconde allumée et 1 seconde éteinte.

Renvoie

void

2.14.2.2 voyant_blink_defaut()

```
void voyant_blink_defaut ( )
```

Cette fonction fait clignoter la led de défaut.

Dans cette fonction on fait clignoter la led de défaut en ROUGE 4 fois avec un intervalle d'une seconde Ce qui donne 8 secondes de clignotement. 1 seconde allumée et 1 seconde éteinte. *

Renvoie

void

2.14.2.3 voyant_set_charge()

```
void voyant_set_charge (
            led charge )
```

Cette fonction met la led de charge à VERT ou ROUGE ou l'éteint.

Dans cette fonction on accède la mémoire partagée et on met la led à l'état souhaité

Paramètres

in	*charge*	l'état souhaité de la led

Renvoie

void

2.14.2.4 voyant_set_defaut()

```
void voyant_set_defaut (
            led defaut )
```

Cette fonction met la led de défaut à VERT ou ROUGE ou l'éteint.

Dans cette fonction on accède la mémoire partagée et on met la led à l'état souhaité

Paramètres

in	*defaut*	l'état souhaité de la led

Renvoie

void

2.14.2.5 voyant_set_dispo()

```
void voyant_set_dispo (
            led dispo )
```

Cette fonction met la led de disponibilité à VERT ou ROUGE ou l'éteint.

Dans cette fonction on accède la mémoire partagée et on met la led à l'état souhaité

Paramètres

in	*dispo*	l'état souhaité de la led

Renvoie

void

2.14.2.6 voyant_set_prise()

```
void voyant_set_prise (
            led prise )
```

Cette fonction met la led de prise à VERT ou ROUGE ou l'éteint.

Dans cette fonction on accède la mémoire partagée et on met la led à l'état souhaité

Paramètres

in	*prise*	l'état souhaité de la led

Renvoie

void

2.14.2.7 voyant_set_trappe()

```
void voyant_set_trappe (
            led trappe )
```

Cette fonction met la led de trappe à VERT ou ROUGE ou l'éteint.

Dans cette fonction on accède la mémoire partagée et on met la led à l'état souhaité

Paramètres

in	*trappe*	l'état souhaité de la led

Renvoie

void

2.14.3 Documentation des variables

2.14.3.1 io

```
entrees* io
```

variable globale de voyant.c pour la gestion mémoire partagée

— ∗io : pointeur vers la structure des entrées de la mémoire partagée

2.14.3.2 shmid

```
int shmid
```

variable globale de voyant.c

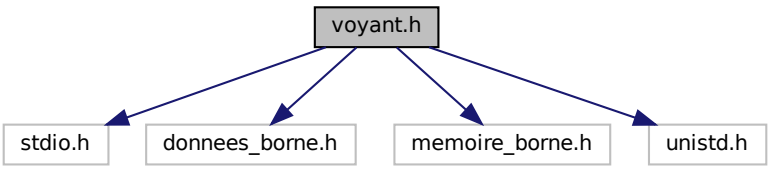
— ∗shmid : entier qui servira d'entrée pour la mémoire partagée

2.15 Référence du fichier voyant.h

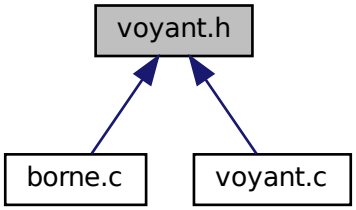
Dans ce fichier on déclare les fonctions pour la gestion des voyants.

```
#include <stdio.h>
#include <donnees_borne.h>
#include <memoire_borne.h>
#include <unistd.h>
```

Graphe des dépendances par inclusion de voyant.h:



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



Fonctions

- void voyant_set_dispo (led dispo)
  Cette fonction met la led de disponibilité à VERT ou ROUGE ou l'éteint.
- void voyant_set_charge (led charge)
  Cette fonction met la led de charge à VERT ou ROUGE ou l'éteint.
- void voyant_blink_charge ()
  Cette fonction fait clignoter la led de charge.
- void voyant_set_defaut (led defaut)
  Cette fonction met la led de défaut à VERT ou ROUGE ou l'éteint.
- void voyant_blink_defaut ()
  Cette fonction fait clignoter la led de défaut.
- void voyant_set_prise (led prise)
  Cette fonction met la led de prise à VERT ou ROUGE ou l'éteint.
- void voyant_set_trappe (led trappe)
  Cette fonction met la led de trappe à VERT ou ROUGE ou l'éteint.

2.15.1 Description détaillée

Dans ce fichier on déclare les fonctions pour la gestion des voyants.

2.15.2 Documentation des fonctions

2.15.2.1 voyant_blink_charge()

```
void voyant_blink_charge ( )
```

Cette fonction fait clignoter la led de charge.

Dans cette fonction on fait clignoter la led de charge en VERT 4 fois avec un intervalle d'une seconde Ce qui donne 8 secondes de clignotement. 1 seconde allumée et 1 seconde éteinte.

Renvoie

void

2.15.2.2 voyant_blink_defaut()

```
void voyant_blink_defaut ( )
```

Cette fonction fait clignoter la led de défaut.

Dans cette fonction on fait clignoter la led de défaut en ROUGE 4 fois avec un intervalle d'une seconde Ce qui donne 8 secondes de clignotement. 1 seconde allumée et 1 seconde éteinte. ∗

Renvoie

void

2.15.2.3 voyant_set_charge()

```
void voyant_set_charge (
            led charge )
```

Cette fonction met la led de charge à VERT ou ROUGE ou l'éteint.

Dans cette fonction on accède la mémoire partagée et on met la led à l'état souhaité

Paramètres

in	*charge*	l'état souhaité de la led

Renvoie

void

2.15.2.4 voyant_set_defaut()

```
void voyant_set_defaut (
            led defaut )
```

Cette fonction met la led de défaut à VERT ou ROUGE ou l'éteint.

Dans cette fonction on accède la mémoire partagée et on met la led à l'état souhaité

Paramètres

in	*defaut*	l'état souhaité de la led

Renvoie

void

2.15.2.5 voyant_set_dispo()

```
void voyant_set_dispo (
            led dispo )
```

Cette fonction met la led de disponibilité à VERT ou ROUGE ou l'éteint.

Dans cette fonction on accède la mémoire partagée et on met la led à l'état souhaité

Paramètres

in	*dispo*	l'état souhaité de la led

Renvoie

void

2.15.2.6 voyant_set_prise()

```
void voyant_set_prise (
             led prise )
```

Cette fonction met la led de prise à VERT ou ROUGE ou l'éteint.

Dans cette fonction on accède la mémoire partagée et on met la led à l'état souhaité

Paramètres

in	*prise*	l'état souhaité de la led

Renvoie

void

2.15.2.7 voyant_set_trappe()

```
void voyant_set_trappe (
             led trappe )
```

Cette fonction met la led de trappe à VERT ou ROUGE ou l'éteint.

Dans cette fonction on accède la mémoire partagée et on met la led à l'état souhaité

Paramètres

in	*trappe*	l'état souhaité de la led

Renvoie

void

Index