MIN17212 - Projet Simulation - Documentation

1.0

Généré par Doxygen 1.8.13

Table des matières

1 Index des classes

1.1 Liste des classes

Liste des classes, structures, unions et interfaces avec une brève description :

2 Index des fichiers

2.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

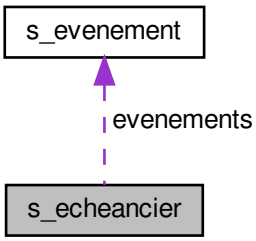
3 Documentation des classes

3.1 Référence de la structure s_echeancier

Représente un ensemble d'évènement en fonction de leur date.

`#include <echeancier.h>`

Graphe de collaboration de s_echeancier :



Attributs publics

- Evenement evenements [MAX_EVENEMENTS]
  Le stockage des événements.
- int n
  Le nombre d'événements actuellement dans l'échéancier.

3.1.1 Description détaillée

Représente un ensemble d'évènement en fonction de leur date.

3.1.2 Documentation des données membres

3.1.2.1 evenements

```
Evenement s_echeancier::evenements[MAX_EVENEMENTS]
```

Le stockage des événements.

3.1.2.2 n

```
int s_echeancier::n
```

Le nombre d'événements actuellement dans l'échéancier.

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

- src/echeancier.h

3.2 Référence de la structure s_evenement

Représente un évènement dans l'échéancier.

`#include <echeancier.h>`

Attributs publics

- TypeEvenement type
  Le type d'événement.
- int k
  L'indentifiant du capteur.
- int etat
  L'état du capteur.
- double date
  La date de l'événement.

3.2.1 Description détaillée

Représente un évènement dans l'échéancier.

3.2.2 Documentation des données membres

3.2.2.1 date

`double s_evenement::date`

La date de l'événement.

3.2.2.2 etat

`int s_evenement::etat`

L'état du capteur.

3.2.2.3 k

`int s_evenement::k`

L'indentifiant du capteur.

3.2.2.4 type

```
TypeEvenement s_evenement::type
```

Le type d'événement.

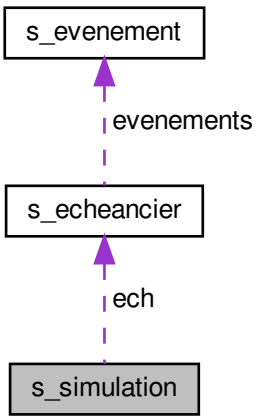
La documentation de cette structure a été générée à partir du fichier suivant :
— src/echeancier.h

3.3 Référence de la structure s_simulation

La structure qui contient les variables de la simulation.

```
#include <lora.h>
```

Graphe de collaboration de s_simulation :



Attributs publics

— Echeancier ech
L'échéancier des évènements.
— int nbTotalEmissions [MAX_K]
Le nombre total d'émissions réussies dans chaque capteur.
— double T
Le temps de simulation.
— int K
Le nombre de capteurs dans le réseau.
— long nbMinEmissions
Nombre de capteurs qui ont atteint le minimum d'émissions réussies.
— long nbCollisions [MAX_ESSAIS]
Nombre de collisions dans chaque état.
— long nbEmissions [MAX_ESSAIS]
Nombre d'émissions totales dans chaque état.
— double tempsEmission [MAX_ESSAIS]
Temps d'émission observée dans chaque état.

3.3.1 Description détaillée

La structure qui contient les variables de la simulation.

3.3.2 Documentation des données membres

3.3.2.1 ech

```
Echeancier s_simulation::ech
```

L'échéancier des évènements.

3.3.2.2 K

```
int s_simulation::K
```

Le nombre de capteurs dans le réseau.

3.3.2.3 nbCollisions

```
long s_simulation::nbCollisions[MAX_ESSAIS]
```

Nombre de collisions dans chaque état.

3.3.2.4 nbEmissions

```
long s_simulation::nbEmissions[MAX_ESSAIS]
```

Nombre d'émissions totales dans chaque état.

3.3.2.5 nbMinEmissions

```
long s_simulation::nbMinEmissions
```

Nombre de capteurs qui ont atteint le minimum d'émissions réussies.

3.3.2.6 nbTotalEmissions

```
int s_simulation::nbTotalEmissions[MAX_K]
```

Le nombre total d'émissions réussies dans chaque capteur.

3.3.2.7 T

```
double s_simulation::T
```

Le temps de simulation.

3.3.2.8 tempsEmission

```
double s_simulation::tempsEmission[MAX_ESSAIS]
```

Temps d'émission observée dans chaque état.

La documentation de cette structure a été générée à partir du fichier suivant :
- src/lora.h

4 Documentation des fichiers

4.1 Référence du fichier src/constantes.h

Macros

- #define LAMBDA_E 10
  Lambda de la durée d'émission d'un paquet.
- #define LAMBDA_I 0.1
  Lambda du temps d'attente après une émission réussie.
- #define LAMBDA_W 0.25
  Lambda du temps d'attente en cas d'échec.
- #define MAX_ESSAIS 7
  Le nombre maximum d'essais d'émission avant que le paquet soit perdu.
- #define MAX_EVENEMENTS 1000
  Le nombre maximum d'évènements dans l'échéancier.
- #define MAX_K 100
  Le nombre maximum de capteurs dans le réseau.
- #define MIN_EMISSIONS 1e3
  Le nombre minimum de messages correctement émis par chaque capteur avant que la simulation soit terminée.

4.1.1 Documentation des macros

4.1.1.1 LAMBDA_E

```
#define LAMBDA_E 10
```

Lambda de la durée d'émission d'un paquet.

4.1.1.2 LAMBDA_I

```
#define LAMBDA_I 0.1
```

Lambda du temps d'attente après une émission réussie.

4.1.1.3 LAMBDA_W

```
#define LAMBDA_W 0.25
```

Lambda du temps d'attente en cas d'échec.

4.1.1.4 MAX_ESSAIS

```
#define MAX_ESSAIS 7
```

Le nombre maximum d'essais d'émission avant que le paquet soit perdu.

4.1.1.5 MAX_EVENEMENTS

```
#define MAX_EVENEMENTS 1000
```

Le nombre maximum d'évènements dans l'échéancier.

4.1.1.6 MAX_K

```
#define MAX_K 100
```

Le nombre maximum de capteurs dans le réseau.

4.1.1.7 MIN_EMISSIONS

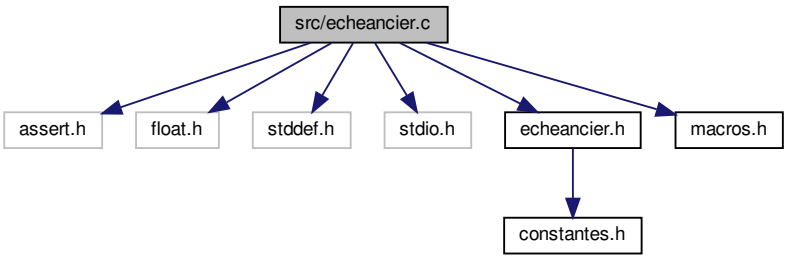
```
#define MIN_EMISSIONS 1e3
```

Le nombre minimum de messages correctement émis par chaque capteur avant que la simulation soit terminée.

4.2 Référence du fichier src/echeancier.c

```
#include <assert.h>
#include <float.h>
#include <stddef.h>
#include <stdio.h>
#include "echeancier.h"
#include "macros.h"
```

Graphe des dépendances par inclusion de echeancier.c :



Fonctions

- void echeancier_ajouter (Echeancier ∗e, TypeEvenement type, int k, int etat, double date)
  Ajoute un événement à l'échéancier.
- Evenement ∗ echeancier_detecter_collision (Echeancier ∗e, double date)
  Détecte si un évènement est en collision avec un autre dans l'échéancier.
- Evenement echeancier_suivant (Echeancier ∗e)
  Récupère le prochain événement de l'échéancier.
- void echeancier_init (Echeancier ∗e)
  Initialise l'échéancier.
- int echeancier_vide (const Echeancier ∗e)
  Vérifie si l'échéancier est vide.

4.2.1 Documentation des fonctions

4.2.1.1 echeancier_ajouter()

```
void echeancier_ajouter (
            Echeancier * e,
            TypeEvenement type,
            int k,
            int etat,
            double date )
```

Ajoute un événement à l'échéancier.

Paramètres

e	L'échéancier.
type	Le type de l'événement.
k	L'indentifiant du capteur.
etat	L'état du capteur.
date	La date de l'événement.

4.2.1.2 echeancier_detecter_collision()

```
Evenement* echeancier_detecter_collision (
            Echeancier * e,
            double date )
```

Détecte si un évènement est en collision avec un autre dans l'échéancier.

Paramètres

e	L'échéancier.
date	La date de l'événement à tester.

Renvoie

L'évènement en collision avec e1, ou NULL si aucun évènement n'est en collision.

4.2.1.3 echeancier_init()

```
void echeancier_init (
            Echeancier * e )
```

Initialise l'échéancier.

Paramètres

e	L'échéancier.

4.2.1.4 echeancier_suivant()

```
Evenement echeancier_suivant (
            Echeancier * e )
```

Récupère le prochain événement de l'échéancier.

Cet évènement est supprimé de l'échéancier. Le comportement est indéfini si l'échéancier est vide.

Paramètres

e	L'échéancier.

Renvoie

Le prochain événement.

Voir également

echeancier_vide

4.2.1.5 echeancier_vide()

```
int echeancier_vide (
            const Echeancier * e )
```

Vérifie si l'échéancier est vide.

Paramètres

e	L'échéancier.

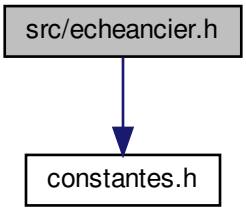
Renvoie

1 si l'échéancier est vide, 0 sinon.

4.3 Référence du fichier src/echeancier.h

```
#include "constantes.h"
```

Graphe des dépendances par inclusion de echeancier.h :



Classes

- struct s_evenement
  Représente un évènement dans l'échéancier.
- struct s_echeancier
  Représente un ensemble d'évènement en fonction de leur date.

Définitions de type

- typedef enum e_type_evenement TypeEvenement
- typedef struct s_evenement Evenement
  Représente un évènement dans l'échéancier.
- typedef struct s_echeancier Echeancier
  Représente un ensemble d'évènement en fonction de leur date.

Énumérations

- enum e_type_evenement { DE, FE, TC }

Fonctions

- void echeancier_ajouter (Echeancier *e, TypeEvenement type, int k, int etat, double date)
  Ajoute un événement à l'échéancier.
- Evenement * echeancier_detecter_collision (Echeancier *e, double date)
  Détecte si un évènement est en collision avec un autre dans l'échéancier.
- Evenement echeancier_suivant (Echeancier *e)
  Récupère le prochain événement de l'échéancier.
- void echeancier_init (Echeancier *e)
  Initialise l'échéancier.
- int echeancier_vide (const Echeancier *e)
  Vérifie si l'échéancier est vide.

4.3.1 Documentation des définitions de type

4.3.1.1 Echeancier

```
typedef struct s_echeancier Echeancier
```

Représente un ensemble d'évènement en fonction de leur date.

4.3.1.2 Evenement

```
typedef struct s_evenement Evenement
```

Représente un évènement dans l'échéancier.

4.3.1.3 TypeEvenement

```
typedef enum e_type_evenement TypeEvenement
```

4.3.2 Documentation du type de l'énumération

4.3.2.1 e_type_evenement

```
enum e_type_evenement
```

Valeurs énumérées

DE	Début émission.
FE	Fin émission.
TC	Traitement collision.

4.3.3 Documentation des fonctions

4.3.3.1 echeancier_ajouter()

```
void echeancier_ajouter (
            Echeancier * e,
            TypeEvenement type,
            int k,
            int etat,
            double date )
```

Ajoute un événement à l'échéancier.

Paramètres

e	L'échéancier.
type	Le type de l'événement.
k	L'indentifiant du capteur.
etat	L'état du capteur.
date	La date de l'événement.

4.3.3.2 echeancier_detecter_collision()

```
Evenement* echeancier_detecter_collision (
            Echeancier * e,
            double date )
```

Détecte si un évènement est en collision avec un autre dans l'échéancier.

Paramètres

e	L'échéancier.
date	La date de l'événement à tester.

Renvoie

L'évènement en collision avec e1, ou NULL si aucun évènement n'est en collision.

4.3.3.3 echeancier_init()

```
void echeancier_init (
            Echeancier * e )
```

Initialise l'échéancier.

Paramètres

e	L'échéancier.

4.3.3.4 echeancier_suivant()

```
Evenement echeancier_suivant (
            Echeancier * e )
```

Récupère le prochain événement de l'échéancier.

Cet évènement est supprimé de l'échéancier. Le comportement est indéfini si l'échéancier est vide.

Paramètres

e	L'échéancier.

Renvoie

Le prochain événement.

Voir également

echeancier_vide

4.3.3.5 echeancier_vide()

```
int echeancier_vide (
            const Echeancier * e )
```

Vérifie si l'échéancier est vide.

Paramètres

e	L'échéancier.

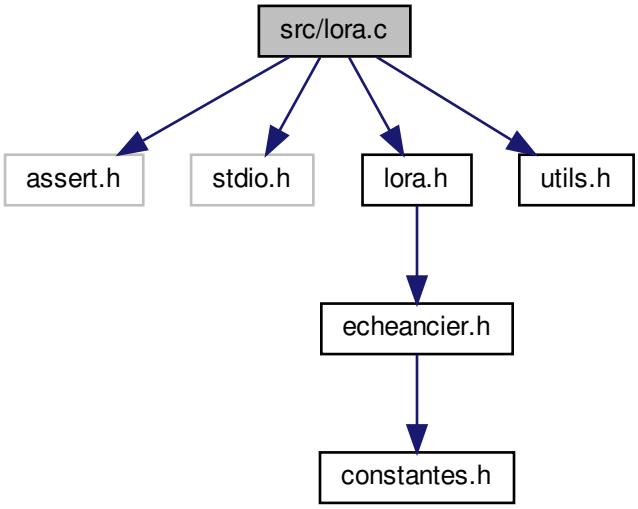
Renvoie

1 si l'échéancier est vide, 0 sinon.

4.4 Référence du fichier src/lora.c

```
#include <assert.h>
#include <stdio.h>
#include "lora.h"
#include "utils.h"
```

Graphe des dépendances par inclusion de lora.c :



Fonctions

- void Traitement_Event (Simulation ∗sim, const Evenement ∗e)
  Traite un événement.
- void Traitement_Collision (Simulation ∗sim, const Evenement ∗e1, Evenement ∗e2)
  Traite une collision.
- void Simulateur (Simulation ∗sim, int showAll)
  Exécute une simulation.
- void simulation_init (Simulation ∗sim, int K)
  Initilise la simulation.
- void simulation_print (const Simulation ∗sim)
  Affiche l'état de la simulation.

4.4.1 Documentation des fonctions

4.4.1.1 Simulateur()

```
void Simulateur (
            Simulation * sim,
            int showAll )
```

Exécute une simulation.

Paramètres

sim	La simulation.
showAll	0 pour n'afficher que la fin de la simulation, sinon affiche toutes les étapes.

4.4.1.2 simulation_init()

```
void simulation_init (
            Simulation * sim,
            int K )
```

Initilise la simulation.

Paramètres

sim	La structure de la simulation à initialiser.
K	Le nombre de capteurs dans le réseau.

4.4.1.3 simulation_print()

```
void simulation_print (
            const Simulation * sim )
```

Affiche l'état de la simulation.

Paramètres

sim	La simulation à afficher.

4.4.1.4 Traitement_Collision()

```
void Traitement_Collision (
            Simulation * sim,
            const Evenement * e1,
            Evenement * e2 )
```

Traite une collision.

Paramètres

sim	La simulation.
e1	L'événement à traiter.
e2	L'évènement à venir en collision.

4.4.1.5 Traitement_Event()

```
void Traitement_Event (
            Simulation * sim,
            const Evenement * e )
```

Traite un événement.

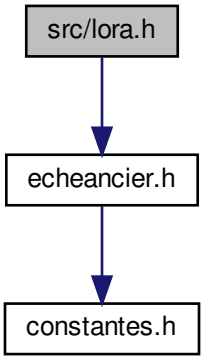
Paramètres

sim	La simulation.
e	L'événement à traiter.

4.5 Référence du fichier src/lora.h

```
#include "echeancier.h"
```

Graphe des dépendances par inclusion de lora.h :



Classes

— struct s_simulation
La structure qui contient les variables de la simulation.

Définitions de type

— typedef struct s_simulation Simulation
La structure qui contient les variables de la simulation.

Fonctions

- void Traitement_Event (Simulation ∗sim, const Evenement ∗e)
  Traite un événement.
- void Traitement_Collision (Simulation ∗sim, const Evenement ∗e1, Evenement ∗e2)
  Traite une collision.
- void Simulateur (Simulation ∗sim, int showAll)
  Exécute une simulation.
- void simulation_init (Simulation ∗sim, int K)
  Initilise la simulation.
- void simulation_print (const Simulation ∗sim)
  Affiche l'état de la simulation.

4.5.1 Documentation des définitions de type

4.5.1.1 Simulation

```
typedef struct s_simulation Simulation
```

La structure qui contient les variables de la simulation.

4.5.2 Documentation des fonctions

4.5.2.1 Simulateur()

```
void Simulateur (
            Simulation * sim,
            int showAll )
```

Exécute une simulation.

Paramètres

sim	La simulation.
showAll	0 pour n'afficher que la fin de la simulation, sinon affiche toutes les étapes.

4.5.2.2 simulation_init()

```
void simulation_init (
            Simulation * sim,
            int K )
```

Initilise la simulation.

Paramètres

sim	La structure de la simulation à initialiser.
K	Le nombre de capteurs dans le réseau.

4.5.2.3 simulation_print()

```
void simulation_print (
            const Simulation * sim )
```

Affiche l'état de la simulation.

Paramètres

sim	La simulation à afficher.

4.5.2.4 Traitement_Collision()

```
void Traitement_Collision (
            Simulation * sim,
            const Evenement * e1,
            Evenement * e2 )
```

Traite une collision.

Paramètres

sim	La simulation.
e1	L'événement à traiter.
e2	L'évènement à venir en collision.

4.5.2.5 Traitement_Event()

```
void Traitement_Event (
            Simulation * sim,
            const Evenement * e )
```

Traite un événement.

Paramètres

sim	La simulation.
e	L'événement à traiter.

4.6 Référence du fichier src/macros.h

Macros

— #define SWAP(A, B)
Echange le contenu de deux variables.

4.6.1 Documentation des macros

4.6.1.1 SWAP

```
#define SWAP(
               A,
               B )
```

Valeur :

```
do { \
        typeof(A) C = (A); \
        (A) = (B); \
        (B) = C; \
    } while (0)
```

Echange le contenu de deux variables.

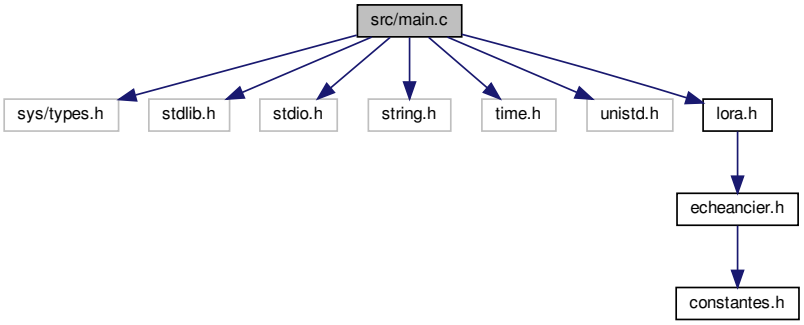
Paramètres

A	La première variable.
B	La seconde variable.

4.7 Référence du fichier src/main.c

```
#include <sys/types.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include "lora.h"
```

Graphe des dépendances par inclusion de main.c :



Fonctions

- static int print_error (const char ∗section, const char ∗message)
- static int print_usage (char ∗bin)
- int main (int ac, char ∗∗av)

4.7.1 Documentation des fonctions

4.7.1.1 main()

```
int main (
            int ac,
            char ** av )
```

4.7.1.2 print_error()

```
static int print_error (
            const char * section,
            const char * message )  [static]
```

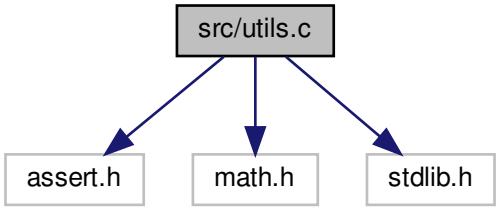
4.7.1.3 print_usage()

```
static int print_usage (
            char * bin )  [static]
```

4.8 Référence du fichier src/utils.c

```
#include <assert.h>
#include <math.h>
#include <stdlib.h>
```

Graphe des dépendances par inclusion de utils.c :



Fonctions

- static double rand_double ()
  Génère un nombre aléatoire entre 0 et 1.
- double Expo_Duree (double lambda)
  Génère un nombre aléatoire suivant une loi exponentielle.

4.8.1 Documentation des fonctions

4.8.1.1 Expo_Duree()

```
double Expo_Duree (
            double lambda )
```

Génère un nombre aléatoire suivant une loi exponentielle.

Le générateur de nombres pseudo-aléatoires doit être initialisé avec srandom().

Paramètres

lambda	La constante de la loi exponentielle.

Renvoie

Le valeur tirée aléatoirement.

4.8.1.2 rand_double()

```
static double rand_double ( )  [static]
```

Génère un nombre aléatoire entre 0 et 1.

Renvoie

Le nombre tirée aléatoirement.

4.9 Référence du fichier src/utils.h

Fonctions

— double Expo_Duree (double lambda)
Génère un nombre aléatoire suivant une loi exponentielle.

4.9.1 Documentation des fonctions

4.9.1.1 Expo_Duree()

```
double Expo_Duree (
            double lambda )
```

Génère un nombre aléatoire suivant une loi exponentielle.

Le générateur de nombres pseudo-aléatoires doit être initialisé avec srandom().

Paramètres

lambda	La constante de la loi exponentielle.

Renvoie

Le valeur tirée aléatoirement.

Index