SIV

1.0

Généré par Doxygen 1.7.6.1

Jeudi Juin 9 2016 21 :41 :18

Table des matières

1 Page principale du projet SIV

1.1 Introduction

Améliorer l'information donnée aux usagers à l'intérieur des bus.

Pour cela, le système devra :
– localiser géographiquement les bus et les arrêts ;
– assurer la communication entre les bus et le poste de commande central (PCC) ;
– informer de manière sonore et visuelle les voyageurs dans le bus ;

1.2 Table des matières

Dépôt SVN : `https://svn.riouxsvn.com/saeiv`

2 Changelog

---

r122 | ccambe | 2016-06-09 18 :20 :18 +0200 (jeu. 09 juin 2016) | 1 ligne

Mise à jour de la simulation avec le gps + avancement du qtp

---

r120 | gdestree | 2016-06-09 08 :45 :59 +0200 (jeu. 09 juin 2016) | 1 ligne

Finitions Classe HautParleur

---

r118 | ccambe | 2016-06-08 16 :07 :21 +0200 (mer. 08 juin 2016) | 1 ligne

Test du simulateur GPS

---

r113 | gdestree | 2016-06-03 16 :59 :26 +0200 (ven. 03 juin 2016) | 1 ligne

Remplacement caractères speciaux pour affichage des arrêts

---

r112 | ccambe | 2016-06-03 16 :23 :20 +0200 (ven. 03 juin 2016) | 1 ligne

Ajout d'affichage divers du qtp (aucunService, priseService, finService etc...), ajout de la méthode finService, mise à jour des courses et service effectuee dans la base de donnees

---

r101 | tvaira | 2016-05-26 06 :28 :38 +0200 (jeu. 26 mai 2016) | 1 ligne

Génération de la documentation avec doxygen

---

r100 | tvaira | 2016-05-26 05 :57 :14 +0200 (jeu. 26 mai 2016) | 1 ligne

Ajout referentiel-sai.sql

---

r98 | tvaira | 2016-05-25 13 :23 :14 +0200 (mer. 25 mai 2016) | 1 ligne

Chemin icone dans Doxyfile

---

r97 | rroux | 2016-05-25 11 :47 :38 +0200 (mer. 25 mai 2016) | 1 ligne

modification de la documentation de la classe TAcquisitionLOcalisation

---

r96 | ccambe | 2016-05-25 09 :42 :52 +0200 (mer. 25 mai 2016) | 1 ligne

Ajout de commentaires pour Doxygen (classe TGestionConducteur et PupitreConducteur)

---

r95 | rroux | 2016-05-24 20 :39 :46 +0200 (mar. 24 mai 2016) | 1 ligne

modification de la documentation des classes gps et tacquisitionlocalisation.

---

r92 | tvaira | 2016-05-23 07 :20 :33 +0200 (lun. 23 mai 2016) | 1 ligne

Mise à jour de la documentation de l'étudiant Destree

---

r91 | gdestree | 2016-05-22 10 :59 :44 +0200 (dim. 22 mai 2016) | 1 ligne

Documentation classe TDiffusionMessages + finition autres classes étudiant n°2

---

r90 | tvaira | 2016-05-21 17 :58 :23 +0200 (sam. 21 mai 2016) | 1 ligne

Documentation classe TInformationVoyageur

---

r88 | tvaira | 2016-05-20 20 :21 :57 +0200 (ven. 20 mai 2016) | 1 ligne

Mise à jour documentation (retrait des classes qextserialport)

---

r87 | tvaira | 2016-05-20 20 :19 :28 +0200 (ven. 20 mai 2016) | 1 ligne

Mise à jour documentation (retrait des classes qextserialport)

---

r82 | gdestree | 2016-05-20 18 :02 :37 +0200 (ven. 20 mai 2016) | 1 ligne

Documentation classe TInformationVoyageur

---

r81 | gdestree | 2016-05-20 12 :31 :03 +0200 (ven. 20 mai 2016) | 1 ligne

---

r79 | tvaira | 2016-05-19 11 :28 :00 +0200 (jeu. 19 mai 2016) | 1 ligne

Ajout des icones pour la documentation

---

r78 | tvaira | 2016-05-19 11 :22 :20 +0200 (jeu. 19 mai 2016) | 1 ligne

Initialisation de la documentation SIV à compléter pour doxygen

---

r72 | gdestree | 2016-05-05 16 :38 :18 +0200 (jeu. 05 mai 2016) | 1 ligne

---

r71 | tvaira | 2016-05-05 09 :13 :59 +0200 (jeu. 05 mai 2016) | 1 ligne

Correction simulateur-gps (SIV)

---

r67 | rroux | 2016-05-04 17 :09 :39 +0200 (mer. 04 mai 2016) | 1 ligne

Mise à jour du simulateur GPS (tv)

---

r66 | ccambe | 2016-05-04 16 :21 :58 +0200 (mer. 04 mai 2016) | 1 ligne

Ajout de chargerService

---

r65 | ccambe | 2016-05-04 14 :53 :59 +0200 (mer. 04 mai 2016) | 1 ligne

Mise à jour de pupitreconducteur.cpp et tgestionconducteur.cpp

---

r64 | rroux | 2016-05-04 14 :37 :35 +0200 (mer. 04 mai 2016) | 1 ligne

mise à jour du code de la classe GPS

---

r63 | gdestree | 2016-05-02 17 :40 :19 +0200 (lun. 02 mai 2016) | 1 ligne

Appropriation Simulation TInformationsVoyageur

---

r62 | tvaira | 2016-05-02 09 :38 :29 +0200 (lun. 02 mai 2016) | 1 ligne

Fin des threads + Simulation TInformationVoyageur (et2)

---

r61 | tvaira | 2016-05-02 09 :20 :56 +0200 (lun. 02 mai 2016) | 1 ligne

Fin des threads + Simulation TInformationVoyageur (et2)

---

r60 | gdestree | 2016-05-01 22 :25 :10 +0200 (dim. 01 mai 2016) | 1 ligne

debug hautparleur

---

r59 | gdestree | 2016-05-01 14 :44 :30 +0200 (dim. 01 mai 2016) | 1 ligne

Finition Simulation d'un itinéraire // TInformationVoyageur()

---

r58 | tvaira | 2016-05-01 13 :45 :36 +0200 (dim. 01 mai 2016) | 1 ligne

Simulation TInformationVoyageur (correction numeroArret)

---

r57 | gdestree | 2016-05-01 12 :22 :05 +0200 (dim. 01 mai 2016) | 1 ligne

Appropriation Simulation TInformationVoyageur + Rajout quelques en-têtes de methodes

---

r56 | tvaira | 2016-05-01 11 :22 :04 +0200 (dim. 01 mai 2016) | 1 ligne

Simulation TInformationVoyageur (ajout simulerEtudiant2())

---

r55 | tvaira | 2016-05-01 11 :01 :34 +0200 (dim. 01 mai 2016) | 1 ligne

Simulation TInformationVoyageur (correction declencherFinArret())

---

r54 | tvaira | 2016-05-01 10 :53 :52 +0200 (dim. 01 mai 2016) | 1 ligne

Simulation TInformationVoyageur

---

r52 | gdestree | 2016-04-30 21 :29 :24 +0200 (sam. 30 avril 2016) | 1 ligne

Mise à jour BUGS.txt + Traitement récupération des arrêts dans la base de donnée (non terminé)

---

r49 | tvaira | 2016-04-29 22 :18 :24 +0200 (ven. 29 avril 2016) | 1 ligne

Panneau : :defiler() à tester \ !

---

r47 | gdestree | 2016-04-29 11 :46 :58 +0200 (ven. 29 avril 2016) | 1 ligne

Correction d'un connect entre la classe TInformationsVoyageur et TDiffusionMessages + Correction en-têtes siv.h

---

r46 | ccambe | 2016-04-29 11 :35 :34 +0200 (ven. 29 avril 2016) | 1 ligne

modification Siv.h --> siv.h dans le siv.pro

---

r45 | gdestree | 2016-04-29 11 :32 :42 +0200 (ven. 29 avril 2016) | 1 ligne

---

r44 | gdestree | 2016-04-28 22 :54 :13 +0200 (jeu. 28 avril 2016) | 1 ligne

Debug

---

r43 | gdestree | 2016-04-28 22 :34 :19 +0200 (jeu. 28 avril 2016) | 1 ligne

Rename siv.h

---

r41 | rroux | 2016-04-28 16 :43 :08 +0200 (jeu. 28 avril 2016) | 1 ligne

ajout des methodes des classes gps et tacquisitionlocalisation et Siv.h

---

r40 | gdestree | 2016-04-28 16 :19 :35 +0200 (jeu. 28 avril 2016) | 1 ligne

Ajout CalculerDistance() et Mise à jour classes TInformationsVoyageur et TDiffussionsMessages

---

r39 | ccambe | 2016-04-28 16 :14 :50 +0200 (jeu. 28 avril 2016) | 1 ligne

Ajout classe prise de service (toujours en cours...)

---

r35 | gdestree | 2016-04-27 10 :00 :07 +0200 (mer. 27 avril 2016) | 1 ligne

Mise à jour TInformationsVoyageur

---

r34 | gdestree | 2016-04-27 09 :00 :54 +0200 (mer. 27 avril 2016) | 1 ligne

Ajout des classes HautParleur et BaseDeDonnees dans le projet .pro

---

r33 | gdestree | 2016-04-27 08 :59 :54 +0200 (mer. 27 avril 2016) | 1 ligne

Ajout des classes HautParleur (vide) et BaseDeDonnees

---

r32 | gdestree | 2016-04-27 08 :57 :28 +0200 (mer. 27 avril 2016) | 1 ligne

Mise à jour de TInformationsVoyageur et ajout de la classe HautParleur

---

r30 | gdestree | 2016-04-26 19 :00 :46 +0200 (mar. 26 avril 2016) | 1 ligne

Ajustements classe TInformationsVoyageur

---

r29 | gdestree | 2016-04-24 18 :39 :31 +0200 (dim. 24 avril 2016) | 1 ligne

Ajout quelques en-têtes

---

r28 | gdestree | 2016-04-23 22 :17 :17 +0200 (sam. 23 avril 2016) | 1 ligne

Démarrage codage Scénario Détecter les points d'arrêts + corrections mineures autres classes

---

r26 | gdestree | 2016-04-22 16 :07 :30 +0200 (ven. 22 avril 2016) | 1 ligne

---

r25 | tvaira | 2016-04-21 18 :36 :31 +0200 (jeu. 21 avril 2016) | 1 ligne

Correction signal/slot entre threads tinformationsvoyageur et tdiffusionmessages

---

r24 | gdestree | 2016-04-21 17 :26 :18 +0200 (jeu. 21 avril 2016) | 1 ligne

Ajout siv.ini + lireParametres() + diffuser()

---

r23 | rroux | 2016-04-21 13 :40 :45 +0200 (jeu. 21 avril 2016) | 1 ligne

déclaration de la classe GPS (méthodes)

---

r21 | gdestree | 2016-04-21 10 :38 :56 +0200 (jeu. 21 avril 2016) | 1 ligne

Squellette classes TinformationsVoyageur et Tdiffussionmessages + Transferts méthodes entre classes Panneau et - Tdiffussionmessages

---

r19 | ccambe | 2016-04-21 10 :15 :52 +0200 (jeu. 21 avril 2016) | 1 ligne

Ajout de commentaires dans pupitreConducteur.cpp

---

r18 | ccambe | 2016-04-21 10 :03 :09 +0200 (jeu. 21 avril 2016) | 1 ligne

ajout de qtp.h et qtp.cpp

---

r17 | ccambe | 2016-04-21 09 :59 :01 +0200 (jeu. 21 avril 2016) | 1 ligne

Menu principal du pupitre conducteur ajouter

---

r14 | gdestree | 2016-03-24 12 :27 :25 +0100 (jeu. 24 mars 2016) | 1 ligne

---

r13 | gdestree | 2016-03-24 11 :34 :14 +0100 (jeu. 24 mars 2016) | 1 ligne

Integration classe Panneau dans thread TDiffusionMessages

---

r11 | gdestree | 2016-03-23 11 :28 :55 +0100 (mer. 23 mars 2016) | 1 ligne

Ajout de la gestion du port serie pour le panneau lumineux

---

r7 | ccambe | 2016-03-23 10 :50 :28 +0100 (mer. 23 mars 2016) | 1 ligne

Ajout commentaire dans Panneau.h (test)

---

r6 | ccambe | 2016-03-23 10 :30 :17 +0100 (mer. 23 mars 2016) | 2 lignes

Ajout de l'architecture logicielle de l'application

---

3 Configuration

`configuration.sh`

Informations sur le poste de développement

- Distribution : Ubuntu 12.04.5 LTS
- OS : GNU/Linux
- Noyau : Linux
- Version : 3.8.0-44-generic
- Machine : x86_64
- Processeur : Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz
- Mémoire RAM : 8129984 kB

Liste des paquets Qt nécessaires

- libqt4-sql libqtgui4 libqtcore4

Liste des autres paquets nécessaires

- libc6 libstdc++6-armel-cross gcc-4.6-arm-linux-gnueabi libc6-armel-cross libfontconfig1 libaudio2 libglib2.0-0 libpng12-0 zlib1g libfreetype6 libglib2.0-0 libsm6 libice6 libxi6 libxrender1 libxext6 libx11-6 libc6-armel-cross libc6-armel-cross libc6-armel-cross libc6 libexpat1 libxt6 libxau6 libpcre3 libffi6 libuuid1 libxcb1 libxdmcp6

génèré le jeudi 19 mai 2016, 07 :30 :44 (UTC+0200)

Informations de version sur les outils

`version.sh`

- g++ (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3
- QMake version 2.01a
- GNU Make 3.81
- GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
- svn, version 1.6.17 (r1128011)
- geany 1.24.1 (construit le May 20 2014 avec GTK 2.24.10, GLib 2.32.4)
- eSpeak text-to-speech : 1.46.02 06.Jan.12 Data at : /usr/share/espeak-data
- sqlite3 3.7.9 2011-11-01 00 :52 :41 c7c6050ef060877ebe77b41d959e9df13f8c9b5e
- Qt Meta Object Compiler version 63 (Qt 4.8.1)
- Qt Creator 2.4.1 based on Qt 4.8.1
- cppunit-config 1.12.1
- doxygen 1.7.6.1

– bouml Bouml 4.23
– papyrus Papyrus 1.1.3

génèré le jeudi 19 mai 2016, 07 :31 :08 (UTC+0200)

4 Manuel d'installation

Configuration de la prise en charge automatique du matériel USB :

$ sudo vim /etc/udev/rules.d/51-ttyusb.rules

SUBSYSTEMS=="usb", ATTRS{idVendor}=="067b", ATTRS{idProduct}=="2303", MODE="0666", SYMLINK+="qtp"

SUBSYSTEMS=="usb", ATTRS{idVendor}=="0557", ATTRS{idProduct}=="2008", MODE="0666", SYMLINK+="panneau"

SUBSYSTEMS=="usb", ATTRS{idVendor}=="05c6", ATTRS{idProduct}=="9000", MODE="0666", SYMLINK+="gps"

Vérification :

$ ls -l /dev/ | grep ttyUSB

lrwxrwxrwx 1 root root 7 avril 21 09 :50 panneau -> ttyUSB1

lrwxrwxrwx 1 root root 7 avril 21 09 :50 qtp -> ttyUSB0

crw-rw-rw- 1 root dialout 188, 0 avril 21 09 :51 ttyUSB0

crw-rw-rw- 1 root dialout 188, 1 avril 21 09 :52 ttyUSB1

Fabrication de l'exécutable :
– qmake
– make

5 Recette

Étudiant 1 : CAMBE Cyril (SIV)

Le menu principal de gestion de service est affiché et fonctionnel

Le Chauffeur peut saisir son code et réaliser la prise de service

La prise de service est enregistrée dans la base de données embarquée

Le Chauffeur peut démarrer une course

Le Chauffeur peut visualiser une course sur son pupitre

Le Chauffeur peut visualiser les informations lors des différents arrêts d'un itinéraire

Le Chauffeur peut mettre fin à son service

La fin de service est enregistrée dans la base de données embarquée

La prise de service et fin de service sont signalées

Étudiant 2 : DESTREE Gabriel (SIV)

L'arrivée sur un arrêt et le départ sont détectés et signalés

Le bandeau lumineux affiche périodiquement les informations à destination des voyageurs (la date et l'heure, la destination, le numéro de ligne, l'arrêt ou le prochain arrêt)

Les haut-parleurs annoncent l'arrêt ou le prochain arrêt

L'écran vidéo affiche la destination, le numéro et le nom de la ligne

L'écran vidéo affiche et actualise le thermomètre (avec au moins 3 arrêts indiqués) du parcours

La fin d'une course est signalée

Étudiant 3 : ROUX Rémy (SIV)

Les trames sont lues périodiquement en provenance du GPS et du simulateur fourni

Les données de géolocalisation (latitude et longitude) sont extraites et décodées

Les données de géolocalisation (latitude et longitude) sont signalées

Le protocole de communication entre un client SIV et le serveur SAI est spécifié et mis en oeuvre

Une communication avec le SAI est possible

6 Base de données

-- -- -- -- Structure de la table ‘organisme‘ --

CREATE TABLE ‘organisme‘ (‘idOrganisme‘ bigint(20) NOT NULL, ‘nomOrganisme‘ varchar(160) DEFAULT NULL, ‘telephone‘ varchar(10) DEFAULT NULL, ‘fuseauHoraire‘ varchar(160) DEFAULT NULL, ‘langue‘ varchar(160) DEFAULT NULL, ‘url‘ varchar(160) DEFAULT NULL, PRIMARY KEY (‘idOrganisme‘)) ;

-- -- Contenu de la table ‘organisme‘ --

INSERT INTO ‘organisme‘ (‘idOrganisme‘, ‘nomOrganisme‘, ‘telephone‘, ‘fuseauHoraire‘, ‘langue‘, ‘url‘) VALUES (6192449487677451, ’Tisséo’, ’0561417070’, ’Europe/Paris’, ’fr’, ’http://www.tisseo.fr’) ;

-- -- -- -- Structure de la table ‘conducteur‘ --

CREATE TABLE ‘conducteur‘ (‘codeConducteur‘ smallint(4) NOT NULL, ‘nomConducteur‘ varchar(160) DEFAULT NULL, PRIMARY KEY (‘codeConducteur‘)) ;

-- -- Contenu de la table ‘conducteur‘ --

INSERT INTO ‘conducteur‘ (‘codeConducteur‘, ‘nomConducteur‘) VALUES (1, ’denis darmon’) ; INSERT INTO ‘conducteur‘ (‘codeConducteur‘, ‘nomConducteur‘) VALUES (195, ’richard boue’) ; INSERT INTO ‘conducteur‘ (‘codeConducteur‘, ‘nomConducteur‘) VALUES (1234, ’fabrice allaire’) ; INSERT INTO ‘conducteur‘ (‘codeConducteur‘, ‘nomConducteur‘) VALUES (1962, ’nicolas lepretre’) ; INSERT INTO ‘conducteur‘ (‘codeConducteur‘, ‘nomConducteur‘) VALUES (1993, ’jean lapierre’) ; INSERT INTO ‘conducteur‘ (‘codeConducteur‘, ‘nomConducteur‘) VALUES (2121, ’cyril salomon’) ; INSERT INTO ‘conducteur‘ (‘codeConducteur‘, ‘nomConducteur‘) VALUES (3082, ’olivier piquet’) ; INSERT INTO ‘conducteur‘ (‘codeConducteur‘, ‘nomConducteur‘) VALUES (3553, ’laurine thibault’) ; INSERT INTO ‘conducteur‘ (‘codeConducteur‘, ‘nomConducteur‘) VALUES (6340, ’marc gross’) ; INSERT INTO ‘conducteur‘ (‘codeConducteur‘, ‘nomConducteur‘) VALUES (7275, ’fanny delaunay’) ;

-- -- -- -- Structure de la table ‘vehicule‘ --

CREATE TABLE ‘vehicule‘ (‘idVehicule‘ smallint(3) NOT NULL, ‘typeVehicule‘ varchar(160) DEFAULT NULL, ‘capacite‘ smallint(3) DEFAULT NULL, PRIMARY KEY (‘idVehicule‘)) ;

-- -- Contenu de la table ‘vehicule‘ --

INSERT INTO ‘vehicule‘ (‘idVehicule‘, ‘typeVehicule‘, ‘capacite‘) VALUES (82, ’articulé’, 148) ; INSERT INTO ‘vehicule‘ (‘idVehicule‘, ‘typeVehicule‘, ‘capacite‘) VALUES (123, ’standard’, 100) ; INSERT INTO ‘vehicule‘ (‘idVehicule‘, ‘typeVehicule‘, ‘capacite‘) VALUES (155, ’standard’, 100) ; INSERT INTO ‘vehicule‘ (‘idVehicule‘, ‘typeVehicule‘, ‘capacite‘) VALUES (171, ’articulé’, 148) ; INSERT INTO ‘vehicule‘ (‘idVehicule‘, ‘typeVehicule‘, ‘capacite‘) VALUES (193, ’standard’, 100) ; INSERT INTO ‘vehicule‘ (‘idVehicule‘, ‘typeVehicule‘, ‘capacite‘) VALUES (204, ’articulé’, 148) ; INSERT INTO ‘vehicule‘ (‘idVehicule‘, ‘typeVehicule‘, ‘capacite‘) VALUES (287, ’midibus’, 88) ; INSERT INTO ‘vehicule‘ (‘idVehicule‘, ‘typeVehicule‘, ‘capacite‘) VALUES (345, ’midibus’, 88) ; INSERT INTO ‘vehicule‘ (‘idVehicule‘, ‘typeVehicule‘, ‘capacite‘) VALUES (615, ’articulé’, 148) ; INSERT INTO ‘vehicule‘ (‘idVehicule‘, ‘typeVehicule‘, ‘capacite‘) VALUES (679, ’midibus’, 88) ; INSERT INTO ‘vehicule‘ (‘idVehicule‘, ‘typeVehicule‘, ‘capacite‘) VALUES (766, ’midibus’, 88) ; INSERT INTO ‘vehicule‘ (‘idVehicule‘, ‘typeVehicule‘, ‘capacite‘) VALUES (986, ’articulé’, 148) ;

-- -- -- -- Structure de la table ‘ligne‘ --

CREATE TABLE ‘ligne‘ (‘idLigne‘ bigint(20) NOT NULL DEFAULT ’0’, ‘nomLong‘ varchar(160) DEFAULT NULL, ‘nomCourt‘ int(11) DEFAULT NULL, ‘description‘ varchar(160) DEFAULT NULL, ‘type‘ varchar(160) DEFAULT NULL, ‘couleurLigne‘ varchar(6) DEFAULT ’000000’, ‘couleurTexte‘ varchar(6) DEFAULT ’000000’, ‘idOrganisme‘ bigint(20) NOT NULL, PRIMARY KEY (‘idLigne‘), CONSTRAINT fk_ligne_idOrganisme FOREIGN KEY (idOrganisme) REFERENCES organisme(idOrganisme)) ;

-- -- Contenu de la table ‘ligne‘ --

INSERT INTO ‘ligne‘ (‘idLigne‘, ‘nomLong‘, ‘nomCourt‘, ‘description‘, ‘type‘, ‘couleurLigne‘, ‘couleurTexte‘, ‘idOrganisme‘) VALUES (11821949021891635, ’Colomiers Gare SNCF / Brax le Château’, 32, ’Ligne Colomiers Gare SNCF / Brax le Château’, ’3’, ’8e4a05’, ’FFFFFF’, 6192449487677451) ;

-- -- -- -- Structure de la table ‘calendrier‘ --

CREATE TABLE ‘calendrier‘ (‘idCalendrier‘ bigint(20) NOT NULL, ‘lundi‘ binary(1) DEFAULT NULL, ‘mardi‘ binary(1) DEFAULT NULL, ‘mercredi‘ binary(1) DEFAULT NULL, ‘jeudi‘ binary(1) DEFAULT NULL, ‘vendredi‘ binary(1) DEFAULT NULL, ‘samedi‘ binary(1) DEFAULT NULL, ‘dimanche‘ binary(1) DEFAULT NULL, ‘dateDebut‘ date DEFAULT NULL, ‘dateFin‘ date DEFAULT NULL, PRIMARY KEY (‘idCalendrier‘)) ;

-- -- Contenu de la table ‘calendrier‘ --

INSERT INTO ‘calendrier‘ (‘idCalendrier‘, ‘lundi‘, ‘mardi‘, ‘mercredi‘, ‘jeudi‘, ‘vendredi‘, ‘samedi‘, ‘dimanche‘, ‘dateDebut‘, ‘dateFin‘) VALUES (4503599631512310, ’1’, ’1’, ’1’, ’1’, ’1’, ’0’, ’0’, ’2015-01-01’, ’2016-12-31’) ; INSERT INTO ‘calendrier‘ (‘idCalendrier‘, ‘lundi‘, ‘mardi‘, ‘mercredi‘, ‘jeudi‘, ‘vendredi‘, ‘samedi‘, ‘dimanche‘, ‘dateDebut‘, ‘dateFin‘) VALUES (4503603925963979, ’0’, ’0’, ’0’, ’0’, ’0’, ’1’, ’0’, ’2015-01-01’, ’2016-12-31’) ;

-- -- -- -- Structure de la table ‘itineraire‘ --

CREATE TABLE ‘itineraire‘ (‘idItineraire‘ bigint(20) NOT NULL, ‘destination‘ varchar(160) DEFAULT NULL, ‘direction‘ binary(1) DEFAULT NULL, ‘idCalendrier‘ bigint(20) DEFAULT NULL, ‘idLigne‘ bigint(20) DEFAULT NULL, PRIMARY KEY (‘idItineraire‘), CONSTRAINT fk_itineraire_idCalendrier FOREIGN KEY (idCalendrier) REFERENCES calendrier(idCalendrier), CONSTRAINT fk_itineraire_idLigne FOREIGN KEY (idLigne) REFERENCES ligne(idLigne)) ;

-- -- Contenu de la table ‘itineraire‘ --

INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603922673401,’-Colomiers Gare SNCF COLOMIERS’,0,4503599631512310,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603922673402,’Colomiers Gare SNCF COLOMIERS’,0,4503599631512310,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603922673403,’Colomiers Gare SNCF COLOMIERS’,0,4503599631512310,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603922673404,’-Colomiers Gare SNCF COLOMIERS’,0,4503599631512310,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603922673405,’Colomiers Gare SNCF COLOMIERS’,0,4503599631512310,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603922673406,’Colomiers Gare SNCF COLOMIERS’,0,4503599631512310,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603922673407,’-Colomiers Gare SNCF COLOMIERS’,0,4503599631512310,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603922673408,’Colomiers Gare SNCF COLOMIERS’,0,4503599631512310,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603922673413,’Brax le Château BRAX’,1,4503599631512310,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603922673414,’Brax le Château BRAX’,1,4503599631512310,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603922673415,’Brax le Château BRAX’,1,4503599631512310,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603922673416,’Brax le Château BRAX’,1,4503599631512310,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603922673417,’Brax le Château BRAX’,1,4503599631512310,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603922673418,’Brax le Château BRAX’,1,4503599631512310,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603922673419,’Brax le Château BRAX’,1,4503599631512310,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603922673420,’Brax le Château BRAX’,1,4503599631512310,11821949021891635) ;

INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603925963979,’-Colomiers Gare SNCF COLOMIERS’,0,4503603925963979,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603925963980,’Colomiers Gare SNCF COLOMIERS’,0,4503603925963979,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603925963981,’Colomiers Gare SNCF COLOMIERS’,0,4503603925963979,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603925963982,’Colomiers Gare SNCF COLOMIERS’,0,4503603925963979,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603925963983,’Brax le Château BRAX’,1,4503603925963979,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603925963984,’Brax le Château BRAX’,1,4503603925963979,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603925963985,’Brax le Château BRAX’,1,4503603925963979,11821949021891635) ; INSERT INTO ‘itineraire‘ (‘idItineraire‘, ‘destination‘, ‘direction‘, ‘idCalendrier‘, ‘idLigne‘) VALUES (4503603925963986,’Brax le Château BRAX’,1,4503603925963979,11821949021891635) ;

-- ---------- -- -- Structure de la table ‘arret‘ --

CREATE TABLE ‘arret‘ (‘idArret‘ bigint(20) NOT NULL, ‘heureArrivee‘ time DEFAULT NULL, ‘heureDepart‘ time DEFAULT NULL, ‘numeroSequence‘ int(11) DEFAULT NULL, ‘informationVoyageur‘ int(11) DEFAULT NULL, ‘idItineraire‘ bigint(20) NOT NULL DEFAULT ’0’, PRIMARY KEY (‘idArret‘,‘idItineraire‘), CONSTRAINT fk_arret_idItineraire FOREIGN KEY (idItineraire) REFERENCES itineraire(idItineraire)) ;

-- ---------- -- -- Structure de la table ‘lieu‘ --

CREATE TABLE ‘lieu‘ (‘idArret‘ bigint(20) NOT NULL, ‘codeArret‘ int(11) NOT NULL, ‘nomLieu‘ varchar(160) DEFAULT NULL, ‘latitude‘ varchar(160) DEFAULT NULL, ‘longitude‘ varchar(160) DEFAULT NULL, ‘typeLieu‘ binary(1) DEFAULT NULL, ‘stationParente‘ bigint(20) NOT NULL, PRIMARY KEY (‘idArret‘)) ;

-- -- Contenu de la table ‘lieu‘ --

INSERT INTO ‘lieu‘ (‘idArret‘, ‘codeArret‘, ‘nomLieu‘, ‘latitude‘, ‘longitude‘, ‘typeLieu‘, ‘stationParente‘) VALUES (3377699722724038,19961,’-Brax le Château’,43.6169,1.2403,0,1970324837185068) ; INSERT INTO ‘lieu‘ (‘idArret‘, ‘codeArret‘, ‘nomLieu‘, ‘latitude‘, ‘longitude‘, ‘typeLieu‘, ‘stationParente‘) VALUES (3377699722724037,19951,’La Chauge’,43.6155,1.25,0,1970324837190345) ; INSERT INTO ‘lieu‘ (‘idArret‘, ‘codeArret‘, ‘nomLieu‘, ‘latitude‘, ‘longitude‘, ‘typeLieu‘, ‘stationParente‘) VALUES (3377699722724036,19941,’-

Château Cru',43.6127,1.26828,0,1970324837188856) ; INSERT INTO 'lieu' ('idArret', 'codeArret', 'nomLieu', 'latitude', 'longitude', 'typeLieu', 'stationParente') VALUES (3377699722724035,19931,'Stade',43.6153,1.27652,0,1970324837185071) ; INSERT INTO 'lieu' ('idArret', 'codeArret', 'nomLieu', 'latitude', 'longitude', 'typeLieu', 'stationParente') VALUES (3377699722724034,19921,'-Basilique',43.6177,1.28334,0,1970324837185072) ; INSERT INTO 'lieu' ('idArret', 'codeArret', 'nomLieu', 'latitude', 'longitude', 'typeLieu', 'stationParente') VALUES (3377699722724033,19911,'Tuilerie',43.6162,1.29006,0,1970324837188855) ; INSERT INTO 'lieu' ('idArret', 'codeArret', 'nomLieu', 'latitude', 'longitude', 'typeLieu', 'stationParente') VALUES (3377699721034355,19901,'-Lycée International',43.6122,1.31031,0,1970324837185074) ; INSERT INTO 'lieu' ('idArret', 'codeArret', 'nomLieu', 'latitude', 'longitude', 'typeLieu', 'stationParente') VALUES (3377699722724032,19891,'Piquemil',43.6123,1.31955,0,1970324837189860) ; INSERT INTO 'lieu' ('idArret', 'codeArret', 'nomLieu', 'latitude', 'longitude', 'typeLieu', 'stationParente') VALUES (3377699720881634,16240,'-Passerelle',43.6117,1.33032,0,1970324837186554) ; INSERT INTO 'lieu' ('idArret', 'codeArret', 'nomLieu', 'latitude', 'longitude', 'typeLieu', 'stationParente') VALUES (3377699721158291,19735,'Colomiers Gare SNCF',43.6042,1.3347,0,1970324837184752) ;

-- ---------- -- -- Structure de la table 'service' --

CREATE TABLE 'service' ('idService' bigint(20) NOT NULL, 'codeConducteur' smallint(4) DEFAULT NULL, 'idVehicule' smallint(3) DEFAULT NULL, 'dateDebut' date DEFAULT NULL, 'heureDebut' time DEFAULT NULL, 'dateFin' date DEFAULT NULL, 'heureFin' time DEFAULT NULL, 'effectue' int(1) NOT NULL DEFAULT 0, PRIMARY KEY ('idService'), CONSTRAINT fk_service-_codeConducteur FOREIGN KEY (codeConducteur) REFERENCES conducteur(codeConducteur), CONSTRAINT fk_service_-idVehicule FOREIGN KEY (idVehicule) REFERENCES vehicule(idVehicule)) ;

-- -- Contenu de la table 'service' --

INSERT INTO 'service' ('idService', 'codeConducteur', 'idVehicule', 'dateDebut', 'heureDebut', 'dateFin', 'heureFin' , 'effectue') VALUES (3, 1993, 123, '2015-07-11', '07 :00 :00', '', '', 0) ; INSERT INTO 'service' ('idService', 'codeConducteur', 'idVehicule', 'dateDebut', 'heureDebut', 'dateFin', 'heureFin' , 'effectue') VALUES (4, 2121, 123, '2015-07-11', '09 :00 :00', '', '', 0) ;

-- ---------- -- -- Structure de la table 'course' --

CREATE TABLE 'course' ('idItineraire' bigint(20) NOT NULL, 'idService' smallint(3) DEFAULT NULL, 'effectue' int(1) NOT NULL DEFAULT 0, PRIMARY KEY ('idItineraire', 'idService'), CONSTRAINT fk_course_idItineraire FOREIGN KEY (idItineraire) REFERENCES itineraire(idItineraire), CONSTRAINT fk_course_idService FOREIGN KEY (idService) REFERENCES service(idService)) ;

-- -- Contenu de la table 'course' --

INSERT INTO 'course' ('idItineraire', 'idService', 'effectue') VALUES (4503603922673401, 3, 0) ; INSERT INTO 'course' ('idItineraire', 'idService', 'effectue') VALUES (4503603922673413, 3, 0) ; INSERT INTO 'course' ('idItineraire', 'idService', 'effectue') VALUES (4503603922673402, 3, 0) ; INSERT INTO 'course' ('idItineraire', 'idService', 'effectue') VALUES (4503603922673414, 3, 0) ; INSERT INTO 'course' ('idItineraire', 'idService', 'effectue') VALUES (4503603922673403, 3, 0) ; INSERT INTO 'course' ('idItineraire', 'idService', 'effectue') VALUES (4503603922673415, 3, 0) ; INSERT INTO 'course' ('idItineraire', 'idService', 'effectue') VALUES (4503603922673404, 3, 0) ; INSERT INTO 'course' ('idItineraire', 'idService', 'effectue') VALUES (4503603922673416, 3, 0) ; INSERT INTO 'course' ('idItineraire', 'idService', 'effectue') VALUES (4503603922673405, 3, 0) ; INSERT INTO 'course' ('idItineraire', 'idService', 'effectue') VALUES (4503603922673417, 3, 0) ;

INSERT INTO 'course' ('idItineraire', 'idService', 'effectue') VALUES (4503603922673406, 4, 0) ; INSERT INTO 'course' ('idItineraire', 'idService', 'effectue') VALUES (4503603922673418, 4, 0) ; INSERT INTO 'course' ('idItineraire', 'idService', 'effectue') VALUES (4503603922673407, 4, 0) ; INSERT INTO 'course' ('idItineraire', 'idService', 'effectue') VALUES (4503603922673419, 4, 0) ; INSERT INTO 'course' ('idItineraire', 'idService', 'effectue') VALUES (4503603922673408, 4, 0) ; INSERT INTO 'course' ('idItineraire', 'idService', 'effectue') VALUES (4503603922673420, 4, 0) ;

7 A propos

Auteur

Cyril Cambe <cambecyril@yahoo.fr>
Gabriel Destrée <gabriel.destree@gmail.com>
Rémy Roux <remy.roux84130@gmail.com>

Version

1.0

Date

2016

8 Licence GPL

This program is free software ; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation ; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY ; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program ; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

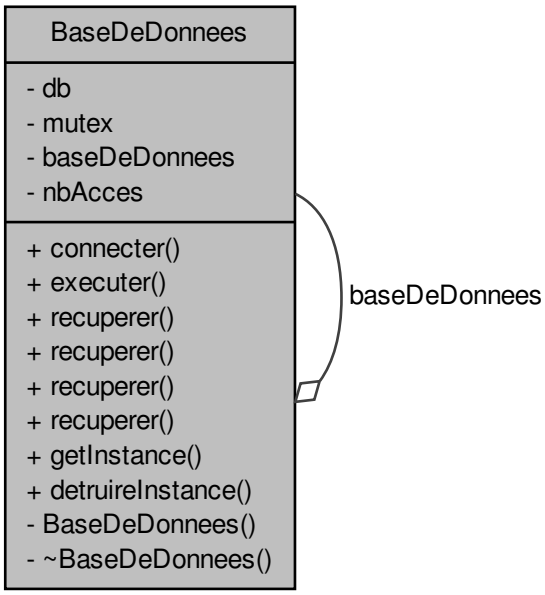
9 Documentation des classes

9.1 Référence de la classe BaseDeDonnees

Classe de gestion de base de données SQLite.

`#include <basededonnees.h>`

Graphe de collaboration de BaseDeDonnees :



Fonctions membres publiques

- bool connecter (QString nomReferentiel="referentiel-sai.sqlite")
  Se connecte à une base de données.
- bool executer (QString requete)
  Exécute une requête SQL UPDATE, INSERT ou DELETE.
- bool recuperer (QString requete, QVector< QString > &donnees)
  Exécute une requête SQL SELECT pour récupérer un seul champ de plusieurs enregistrements : le champ des différents enregistrements est stocké dans un QVector de QString.
- bool recuperer (QString requete, QVector< QStringList > &donnees)
  Exécute une requête SQL SELECT pour récupérer plusieurs champs de plusieurs enregistrements : les différents champs des différents enregistrements sont stockés dans un QVector de QStringList.
- bool recuperer (QString requete, QString &donnees)
  Exécute une requête SQL SELECT pour récupérer un seul champ d'un seul enregistrement : le champ est stocké dans un QString.
- bool recuperer (QString requete, QStringList &donnees)

Exécute une requête SQL SELECT pour récupérer plusieurs champs d'un seul enregistrement : les différents champs sont stockés dans un QStringList.

Fonctions membres publiques statiques

- static BaseDeDonnees * getInstance ()
  Retourne l'unique instance (membre statique)
- static void detruireInstance ()
  Libère l'instance obtenue.

Fonctions membres privées

- BaseDeDonnees ()
  Constructeur placé en privé pour interdire la création d'une instance de l'extérieur de la classe.
- ~BaseDeDonnees ()
  Destructeur.

Attributs privés

- QSqlDatabase db
- QMutex mutex

Attributs privés statiques

- static BaseDeDonnees * baseDeDonnees = NULL
- static int nbAcces = 0

9.1.1 Description détaillée

Classe singleton + ressource critique protégée par un mutex

Avertissement

Il faut ajouter QT += sql au fichier .pro

Auteur

Thierry Vaira <tvaira@free.fr>

9.1.2 Documentation des constructeurs et destructeur

9.1.2.1 BaseDeDonnees : :BaseDeDonnees ( ) [private]

Références db.

Référencé par getInstance().

```
{
    #ifdef DEBUG_BASEDEDONNEES_1
    qDebug() << "<BaseDeDonnees::BaseDeDonnees()>";
    #endif
    db = QSqlDatabase::addDatabase("QSQLITE");
}
```

9.1.2.2 BaseDeDonnees : :~BaseDeDonnees ( ) [private]

```
{
    #ifdef DEBUG_BASEDEDONNEES_1
    qDebug() << "<BaseDeDonnees::~BaseDeDonnees()>";
    #endif
}
```

9.1.3 Documentation des fonctions membres

9.1.3.1 BaseDeDonnees : :connecter (QString *nomReferentiel* = "referentiel-sai.sqlite")

Paramètres

nomReferentiel	QString nom du fichier représentant la base de données (par défaut referentiel-sai.sqlite)

Renvoie

bool vrai si la connexion a réussi sinon faux

Références db, et mutex.

Référencé par PupitreConducteur : :setIdVehicule(), TGestionConducteur : :TGestionConducteur(), et TInformationVoyageur : :TInformationVoyageur().

```
{
    QMutexLocker verrou(&mutex);

    if(db.isOpen())
    {
        #ifdef DEBUG_BASEDEDONNEES
        qDebug() << QString::fromUtf8("<BaseDeDonnees::connecter()> connexion
       active");
        #endif

        return true;
    }

    db.setDatabaseName(QCoreApplication::applicationDirPath() + "/referentiel/"
       + nomReferentiel);
    db.open();
    if(!db.isOpen())
    {
        //qDebug() << db.lastError().text();
        qDebug() << QString::fromUtf8("<BaseDeDonnees::connecter()> erreur :
       impossible de se connecter à la base de données %1 !").arg(
      QCoreApplication::applicationDirPath() + "/referentiel/" + nomReferentiel);

        //QMessageBox::critical(0, QString::fromUtf8("SIV"),
       QString::fromUtf8("Impossible de se connecter à la base de données !"));

        return false;
    }

    #ifdef DEBUG_BASEDEDONNEES
    qDebug() << QString::fromUtf8("<BaseDeDonnees::connecter()> connexion
       réussie");
    #endif

    return true;
}
```

9.1.3.2 BaseDeDonnees : :detruireInstance ( ) [static]

Références baseDeDonnees, et nbActes.

Référencé par TAcquisitionLocalisation : :~TAcquisitionLocalisation(), et TInformationVoyageur : :~TInformationVoyageur().

```
{
    // instance ?
    if(baseDeDonnees != NULL)
    {
        nbActes--;
        #ifdef DEBUG_BASEDEDONNEES_1
        qDebug() << "<BaseDeDonnees::detruireInstance()> nbActes restants = " <
      < nbActes;
        #endif
        // dernier ?
        if(nbActes == 0)
            delete baseDeDonnees;
    }
}
```

9.1.3.3 BaseDeDonnees : :executer (QString *requete*)

Paramètres

requete	QString une requête SQL UPDATE, INSERT ou DELETE

Renvoie

bool vrai si la requête a été exécutée avec succès sinon faux

Références db, et mutex.

Référencé par TGestionConducteur : :chargerService(), TGestionConducteur : :terminerCourse(), et TGestionConducteur-: :terminerService().

```
{
    QMutexLocker verrou(&mutex);
    QSqlQuery r;
    bool retour;

    if(db.isOpen())
    {
        retour = r.exec(requete);
        #ifdef DEBUG_BASEDEDONNEES
        qDebug() << QString::fromUtf8("<BaseDeDonnees::executer()> retour %1
       pour la requete : %2").arg(QString::number(retour)).arg(requete);
        #endif
        if(retour)
        {
            return true;
        }
        else
        {
            qDebug() << QString::fromUtf8("<BaseDeDonnees::executer()> erreur :
       %1 pour la requête %2").arg(r.lastError().text()).arg(requete);

            return false;
        }
    }
    else
        return false;

}
```

9.1.3.4 BaseDeDonnees : :getInstance ( ) [static]

Renvoie

BaseDeDonnees

Références BaseDeDonnees(), baseDeDonnees, et nbAcces.

Référencé par PupitreConducteur : :PupitreConducteur(), TAcquisitionLocalisation : :TAcquisitionLocalisation(), TGestion-Conducteur : :TGestionConducteur(), et TInformationVoyageur : :TInformationVoyageur().

```
{
    if(baseDeDonnees == NULL)
        baseDeDonnees = new BaseDeDonnees();

    nbAcces++;
    #ifdef DEBUG_BASEDEDONNEES_1
    qDebug() << "<BaseDeDonnees::getInstance()> nbAcces = " << nbAcces;
    #endif

    return baseDeDonnees;
}
```

9.1.3.5 BaseDeDonnees : :recuperer (QString *requete,* QVector< QString > & *donnees*)

Paramètres

requete	QString
donnees	QVector<QString>

Renvoie

bool vrai si la requête a été exécutée avec succès sinon faux

Références db, et mutex.

Référencé par TGestionConducteur : :chargerCourses(), TInformationVoyageur : :chargerItineraire(), TGestionConducteur-: :chargerService(), PupitreConducteur : :verifierIdentifiantConducteur(), et PupitreConducteur : :visualiserInformationsArret().

```
{
```

```
    QMutexLocker verrou(&mutex);
    QSqlQuery r;
    bool retour;
    QString data;

    if(db.isOpen())
    {
        retour = r.exec(requete);
        #ifdef DEBUG_BASEDEDONNEES
        qDebug() << QString::fromUtf8("<BaseDeDonnees::recuperer(QString,
       QVector<QString>)> retour %1 pour la requete : %2").arg(QString::number(retour)).arg(
      requete);
        #endif
        if(retour)
        {
            // pour chaque enregistrement
            while ( r.next() )
            {
                // on récupère sous forme de QString la valeur du champs
       sélectionné
                data = r.value(0).toString();

                #ifdef DEBUG_BASEDEDONNEES
                //qDebug() << "<BaseDeDonnees::recuperer(QString,
       QVector<QString>)> enregistrement -> " << data;
                #endif

                // on stocke l'enregistrement dans le QVector
                donnees.push_back(data);
            }
            #ifdef DEBUG_BASEDEDONNEES
            qDebug() << "<BaseDeDonnees::recuperer(QString, QVector<QString>)>
       enregistrement -> " << donnees;
            #endif
            return true;
        }
        else
        {
            qDebug() << QString::fromUtf8("<BaseDeDonnees::recuperer(QString,
       QVector<QString>)> erreur : %1 pour la requête %2").arg(r.lastError().text()).arg
      (requete);

            return false;
        }
    }
    else
        return false;
}
```

9.1.3.6 BaseDeDonnees : :recuperer (QString *requete,* QVector< QStringList > & *donnees*)

Paramètres

requete	QString
donnees	QVector<QStringList>

Renvoie

bool vrai si la requête a été exécutée avec succès sinon faux

Références db, et mutex.

```
{
    QMutexLocker verrou(&mutex);
    QSqlQuery r;
    bool retour;
    QStringList data;

    if(db.isOpen())
    {
        retour = r.exec(requete);
        #ifdef DEBUG_BASEDEDONNEES
        qDebug() << QString::fromUtf8("<BaseDeDonnees::recuperer(QString,
       QVector<QStringList>)> retour %1 pour la requete : %2").arg(QString::number(retour)).
      arg(requete);
        #endif
        if(retour)
        {
            // pour chaque enregistrement
            while ( r.next() )
            {
                // on récupère sous forme de QString la valeur de tous les
       champs sélectionnés
                // et on les stocke dans une liste de QString
                for(int i=0;i<r.record().count();i++)
```

```
                data << r.value(i).toString();

            #ifdef DEBUG_BASEDEDONNEES
            //qDebug() << "<BaseDeDonnees::recuperer(QString,
    QVector<QStringList>)> enregistrement -> " << data;
            /*for(int i=0;i<r.record().count();i++)
                qDebug() << r.value(i).toString();*/
            #endif

            // on stocke l'enregistrement dans le QVector
            donnees.push_back(data);

            // on efface la liste de QString pour le prochain
    enregistrement
            data.clear();
        }
        #ifdef DEBUG_BASEDEDONNEES
        qDebug() << "<BaseDeDonnees::recuperer(QString,
    QVector<QStringList>)> enregistrement -> " << donnees;
        #endif
        return true;
     }
     else
     {
        qDebug() << QString::fromUtf8("<BaseDeDonnees::recuperer(QString,
    QVector<QStringList>)> erreur : %1 pour la requête %2").arg(r.lastError().text())
   .arg(requete);

        return false;
     }
  }
  else
      return false;
}
```

9.1.3.7 BaseDeDonnees : :recuperer (QString *requete,* QString & *donnees*)

Paramètres

requete	QString
donnees	QString

Renvoie

bool vrai si la requête a été exécutée avec succès sinon faux

Références db, et mutex.

```
{
    QMutexLocker verrou(&mutex);
    QSqlQuery r;
    bool retour;

    if(db.isOpen())
    {
        retour = r.exec(requete);
        #ifdef DEBUG_BASEDEDONNEES
        qDebug() << QString::fromUtf8("<BaseDeDonnees::recuperer(QString,
    QString)> retour %1 pour la requete : %2").arg(QString::number(retour)).arg(requete);
        #endif
        if(retour)
        {
            // on se positionne sur l'enregistrement
            r.first();

            // on vérifie l'état de l'enregistrement retourné
            if(!r.isValid())
            {
                #ifdef DEBUG_BASEDEDONNEES
                qDebug() << QString::fromUtf8("
  <BaseDeDonnees::recuperer(QString, QString)> résultat non valide !");
                #endif
                return false;
            }

            // on récupère sous forme de QString la valeur du champ
            if(r.isNull(0))
            {
                #ifdef DEBUG_BASEDEDONNEES
                qDebug() << QString::fromUtf8("
  <BaseDeDonnees::recuperer(QString, QString)> résultat vide !");
                #endif
                return false;
            }
            donnees = r.value(0).toString();
```

```
            #ifdef DEBUG_BASEDEDONNEES
            qDebug() << "<BaseDeDonnees::recuperer(QString, QString)>
       enregistrement -> " << donnees;
            #endif
            return true;
        }
        else
        {
            qDebug() << QString::fromUtf8("<BaseDeDonnees::recuperer(QString,
       QString)> erreur : %1 pour la requête %2").arg(r.lastError().text()).arg(requete)
      ;

            return false;
        }
    }
    else
        return false;
}
```

9.1.3.8 BaseDeDonnees : :recuperer (QString *requete*, QStringList & *donnees*)

Paramètres

requete	QString
donnees	QStringList

Renvoie

bool vrai si la requête a été exécutée avec succès sinon faux

Références db, et mutex.

```
{
    QMutexLocker verrou(&mutex);
    QSqlQuery r;
    bool retour;

    if(db.isOpen())
    {
        retour = r.exec(requete);
        #ifdef DEBUG_BASEDEDONNEES
        qDebug() << QString::fromUtf8("<BaseDeDonnees::recuperer(QString,
       QStringList)> retour %1 pour la requete : %2").arg(QString::number(retour)).arg(
      requete);
        #endif
        if(retour)
        {
            // on se positionne sur l'enregistrement
            r.first();

            // on vérifie l'état de l'enregistrement retourné
            if(!r.isValid())
            {
                #ifdef DEBUG_BASEDEDONNEES
                qDebug() << QString::fromUtf8("
      <BaseDeDonnees::recuperer(QString, QStringList)> résultat non valide !");
                #endif
                return false;
            }

            // on récupère sous forme de QString la valeur de tous les champs
       sélectionnés
            // et on les stocke dans une liste de QString
            for(int i=0;i<r.record().count();i++)
                if(!r.isNull(i))
                    donnees << r.value(i).toString();
            #ifdef DEBUG_BASEDEDONNEES
            qDebug() << "<BaseDeDonnees::recuperer(QString, QStringList)>
       enregistrement -> " << donnees;
            #endif
            return true;
        }
        else
        {
            qDebug() << QString::fromUtf8("<BaseDeDonnees::recuperer(QString,
       QStringList)> erreur : %1 pour la requête %2").arg(r.lastError().text()).arg(
      requete);
            return false;
        }
    }
    else
        return false;
}
```

9.1.4 Documentation des données membres

9.1.4.1 BaseDeDonnees * BaseDeDonnees : :baseDeDonnees = NULL [static, private]

pointeur sur l'instance de la classe (membre statique)

Référencé par detruireInstance(), et getInstance().

9.1.4.2 QSqlDatabase BaseDeDonnees : :db [private]

un objet QSqlDatabase

Référencé par BaseDeDonnees(), connecter(), executer(), et recuperer().

9.1.4.3 QMutex BaseDeDonnees : :mutex [private]

un mutex pour protéger des accès concurrents à la base de données

Référencé par connecter(), executer(), et recuperer().

9.1.4.4 int BaseDeDonnees : :nbAcces = 0 [static, private]

nombre d'objets en cours d'utilisation de l'instance de la classe

Référencé par detruireInstance(), et getInstance().

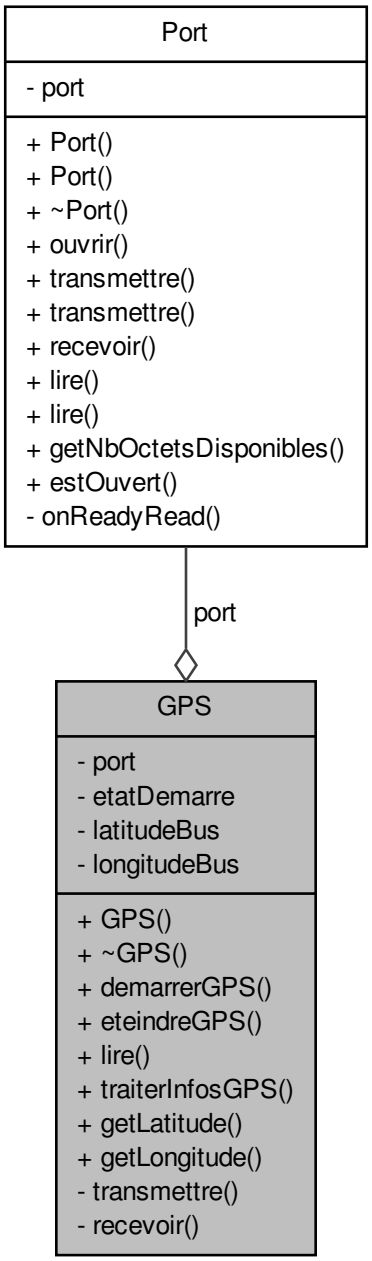
La documentation de cette classe a été générée à partir des fichiers suivants :
– basededonnees.h
– basededonnees.cpp

9.2 Référence de la classe GPS

Classe qui permet l'aquisition et le traitement des données de géolocalisation.

`#include <gps.h>`

Graphe de collaboration de GPS :



Fonctions membres publiques

- GPS ()
  Constructeur par défaut de la classe GPS.
- ~GPS ()
  Destructeur par défaut de la classe GPS.
- bool demarrerGPS ()
  Méthode qui permet de démarrer le GPS.
- bool eteindreGPS ()
  Méthode qui permet d'éteindre le GPS.
- QStringList lire ()
  Méthode qui permet de lire et de découper la trame gps.

- void traiterInfosGPS (QStringList infos)
  Méthode qui permet d'extraire la longitude et la latitude de la trame et de convertir la longitude et la latitude en données exploitables.
- QString getLatitude ()
  Accesseur get qui permet de récuper la latitude.
- QString getLongitude ()
  Accesseur get qui permet de récuper la longitude.

Fonctions membres privées

- int transmettre (const QString &message)
  Méthode qui permet de transmettre des commandes AT au module sim5218e.
- QString recevoir (unsigned int timeout)
  Méthode qui permet de recevoir les réponses du module sim5218e.

Attributs privés

- Port * port
- bool etatDemarre
- QString latitudeBus
- QString longitudeBus

9.2.1 Description détaillée

Auteur

Rémy Roux <remy.roux84130@gmail.com>

9.2.2 Documentation des constructeurs et destructeur

9.2.2.1 GPS : :GPS ( )

Références port.

```
        : port(NULL), etatDemarre(false), latitudeBus(""), longitudeBus("")
{
    // GPS/GPRS sim 5218
    port = new Port(QLatin1String("/dev/ttyUSB2"));
    // Simualteur GPS
    //port = new Port(QLatin1String("/dev/ttyUSB1"));
#ifdef DEBUG_GPS
    qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
#endif
}
```

9.2.2.2 GPS : :~GPS ( )

Références port.

```
{
    delete port;
#ifdef DEBUG_GPS
    qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
#endif
}
```

9.2.3 Documentation des fonctions membres

9.2.3.1 GPS : :demarrerGPS ( )

Renvoie

bool

Références etatDemarre, recevoir(), et transmettre().

Référencé par TAcquisitionLocalisation : :demarrer(), et lire().

```
{
    if(etatDemarre == true)
        return etatDemarre;

    if(transmettre("AT+CGPS=1,1"))
    {
        QString reponse = recevoir(100);
        if(reponse.length() == 0)
            return "";
        if(!reponse.contains("OK"))
            return "";
        reponse.remove(0, 2); // enleve \r\n au début
        etatDemarre = true;
    }

    return etatDemarre;
}
```

9.2.3.2 GPS : :eteindreGPS ( )

Renvoie

bool

Références etatDemarre, recevoir(), et transmettre().

Référencé par TAcquisitionLocalisation : :arreterCourse(), et TAcquisitionLocalisation : :main().

```
{
    if(etatDemarre == false)
        return etatDemarre;

    if(transmettre("AT+CGPS=0,1"))
    {
        QString reponse = recevoir(100);
        if(reponse.length() == 0)
            return "";
        if(!reponse.contains("OK"))
            return "";
        reponse.remove(0, 2); // enleve \r\n au début
        etatDemarre = false;
    }

    return etatDemarre;
}
```

9.2.3.3 GPS : :getLatitude ( )

Renvoie

QString

Références latitudeBus.

Référencé par TAcquisitionLocalisation : :signalerLocalisation().

```
{
    return latitudeBus;
}
```

9.2.3.4 GPS : :getLongitude ( )

Renvoie

QString

Références longitudeBus.

Référencé par TAcquisitionLocalisation : :signalerLocalisation().

```
{
    return longitudeBus;
}
```

9.2.3.5 GPS : :lire ( )

Renvoie

QStringList

Références demarrerGPS(), etatDemarre, recevoir(), et transmettre().

Référencé par TAcquisitionLocalisation : :main().

```
{
    #ifdef DEBUG_GPS
    qDebug() << Q_FUNC_INFO << __LINE__ << QString::fromUtf8("état GPS :") <<
      etatDemarre;
    #endif

    if(etatDemarre == false)
        demarrerGPS();

    if(etatDemarre == false)
        return QStringList("");

    if(transmettre("AT+CGPSINFO"))
    {
        QString reponse = recevoir(100);
        //QString reponse =
       "AT+CGPSINFO\r\r\n+CGPSINFO:3113.343286,N,12121.234064,E,250311,072809.3,44.1,0.0,0\r\nOK\r\n";
        //QString reponse =
       "AT+CGPSINFO\r\r\n+CGPSINFO:4356.925890,N,00448.776416,E,040516,112846.0,90.0,0,0\r\nOK\r\n";
        if(reponse.length() == 0)
        {
            //etatDemarre = false;
            return QStringList("");
        }
        if(!reponse.contains("OK"))
        {
            //etatDemarre = false;
            return QStringList("");
        }

        // Trame reponse :
       AT+CGPSINFO\r\r\n+CGPSINFO:4356.925890,N,00448.776416,E,040516,112846.0,90.0,0,0\r\nOK\r\n
        QStringList donnees = reponse.split('\r');
        int position = -1;
        for(int i = 0; i < donnees.size(); i++)
        {
            if(donnees.at(i).contains("CGPSINFO"))
                position = i;
        }
        if(position == -1)
            return QStringList("");

        // Avant :
       \n+CGPSINFO:4356.925890,N,00448.776416,E,040516,112846.0,90.0,0,0
        QString gpsinfos = donnees[position].remove(0, 11);

#ifdef DEBUG_GPS
        qDebug() << Q_FUNC_INFO << __LINE__ << donnees;
        qDebug() << Q_FUNC_INFO << __LINE__ << gpsinfos;
#endif
        // Après : 4356.925890,N,00448.776416,E,040516,112846.0,90.0,0,0

        //
      +CGPSINFO:[<lat>],[<N/S>],[<log>],[<E/W>],[<date>],[<UTCtime>],[<alt>],[<speed>],[<course>]
        /*   <lat>       Output format is ddmm.mmmm
         *   <log>       Output format is dddmm.mmmm
         *   <date>      Output format is ddmmyy
         *   <UTC time>  Output format is hhmmss.s
         *   <alt>       Unit is meters.
         *   <speed>     Unit is knots.
         *   <course>    Degrees.
         *
         * Exemple :
       +CGPSINFO:4356.925890,N,00448.776416,E,040516,112846.0,90.0,0,0
         */

        QStringList infos = gpsinfos.split(",");

#ifdef DEBUG_GPS
qDebug() << Q_FUNC_INFO << __LINE__ << infos;
#endif
        return infos;
    }

    return QStringList("");
}
```

9.2.3.6 GPS : :recevoir (unsigned int *timeout*) [private]

Paramètres

timeout	

Renvoie

QString

Références Port : :estOuvert(), Port : :getNbOctetsDisponibles(), Port : :lire(), et port.

Référencé par demarrerGPS(), eteindreGPS(), et lire().

```
{
    QString message = "";
    char buffer[1024+1]; // pour le fin de chaine
    int  nbOctets = 0;
    QTime t;
    int debut = 0;

    t.start();

    if (!port->estOuvert()) return "";

    if(timeout != 0)
    {
        debut = t.elapsed();
        while((nbOctets == 0) && (t.elapsed() < (int)(debut + timeout)))
        {
            nbOctets = port->getNbOctetsDisponibles();
            if(nbOctets > 1024)
                nbOctets = 1024;
            if(nbOctets > 0)
            {
                int i = port->lire(buffer, nbOctets);
                if (i != -1)
                {
                    if (buffer[i-1] == '\n' && buffer[i-2] == '\r')
                        buffer[i-2] = '\0';
                    else
                        buffer[i] = '\0';
                }
                else
                    buffer[0] = '\0';
                message += QLatin1String(buffer);
                nbOctets = 0;
            }
        }
#ifdef DEBUG_GPS
        if(message.length() == 0)
            qDebug() << Q_FUNC_INFO << QString::fromUtf8("aucune donnée reçue")
      ;
        else
            qDebug() << Q_FUNC_INFO << QString::fromUtf8("donnée reçue :") <<
      message.toAscii().data();
            //qDebug("%s -> receive (%d) : %s", Q_FUNC_INFO, message.length(),
       message.toAscii().data());
        //qDebug("en %d ms", t.elapsed());
#endif

        return message;
    }
    else
    {
        while(nbOctets == 0)
        {
            nbOctets = port->getNbOctetsDisponibles();
            if(nbOctets > 1024)
                nbOctets = 1024;
        }
    }

    int i = port->lire(/*buffer, nbOctets*/);
    if (i != -1)
    {
        if (buffer[i-1] == '\n' && buffer[i-2] == '\r')
            buffer[i-2] = '\0';
        else
            buffer[i] = '\0';
    }
    else
        buffer[0] = '\0';
    message = QLatin1String(buffer);

#ifdef DEBUG_GPS
    qDebug("-> receive (%d/%d) : %s", i, nbOctets, message.toAscii().data());
#endif
```

```
    return message;
}
```

9.2.3.7 GPS : :traiterInfosGPS (QStringList *infos*)

Paramètres

infos	

Références latitudeBus, et longitudeBus.

Référencé par TAcquisitionLocalisation : :main().

```
{
#ifdef DEBUG_GPS
    qDebug() << Q_FUNC_INFO << __LINE__;
    if(infos.size() > 0)
        qDebug() << "<lat> : " << infos.at(0);
    if(infos.size() > 1)
        qDebug() << "<N/S> : " << infos.at(1);
    if(infos.size() > 2)
        qDebug() << "<log> : " << infos.at(2);
    if(infos.size() > 3)
        qDebug() << "<E/W> : " << infos.at(3);
    if(infos.size() > 4)
        qDebug() << "<date> : " << infos.at(4);
    if(infos.size() > 5)
        qDebug() << "<UTC time> : " << infos.at(5);
    if(infos.size() > 6)
        qDebug() << "<alt> : " << infos.at(6);
    if(infos.size() > 7)
        qDebug() << "<speed> : " << infos.at(7);
    if(infos.size() > 8)
        qDebug() << "<course> : " << infos.at(8);
    if(infos.size() == 0)
        qDebug() << "Aucunes donnees !";
#endif
    QString latitudeD;
    QString latitudeM;
    QString longitudeD;
    QString longitudeM;

    if(infos.size() > 0)
        latitudeD = infos.at(0);
    if(infos.size() > 0)
        latitudeM = infos.at(0);
    if(infos.size() > 2)
        longitudeD = infos.at(2);
    if(infos.size() > 2)
        longitudeM = infos.at(2);

    if(latitudeD.isEmpty() || latitudeM.isEmpty() || longitudeD.isEmpty() ||
      longitudeM.isEmpty())
        return;

#ifdef DEBUG_GPS
    qDebug() << Q_FUNC_INFO << __LINE__ << latitudeD << longitudeD;
#endif

    // Latitude :
    // Conversion format GPS ddmm.mmmm en degré décimal dd.dddd
    // dd + mm.mmmm/60
    // 4356.925890 ---> 43 + (56.925890/60) = 43.94876483 degre

    bool ok = false;
    double latitudeDDouble;
    double latitudeMDouble;
    double latitudeDouble;
    double longitudeDDouble;
    double longitudeMDouble;
    double longitudeDouble;
    QString longitude;
    QString latitude;

    // Pour la latitude
    // 1. Extraire dans le format ddmm.mmmm -> dd
    latitudeD.remove(2, 9);

#ifdef DEBUG_GPS
    //qDebug() << Q_FUNC_INFO << __LINE__ << latitudeD;
#endif

    // 2. Extraire dans le format ddmm.mmmm -> mm.mmmm
    latitudeM.remove(0, 2);

#ifdef DEBUG_GPS
```

```
    //qDebug() << Q_FUNC_INFO << __LINE__ << latitudeM;
#endif

    // 3. Convertir dd -> double
    latitudeDDouble = latitudeD.toDouble(&ok);

#ifdef DEBUG_GPS
    //qDebug() << Q_FUNC_INFO << __LINE__ << latitudeDDouble;
#endif

    // 4. Convertir mm.mmmm -> double
    latitudeMDouble = latitudeM.toDouble(&ok);

#ifdef DEBUG_GPS
    //qDebug() << Q_FUNC_INFO << __LINE__ << latitudeMDouble;
#endif

    // 5. Faire : 43 + (56.925890/60) = 43.94876483
    latitudeDouble = latitudeDDouble + (latitudeMDouble/60.0);

#ifdef DEBUG_GPS
    //qDebug() << Q_FUNC_INFO << __LINE__ << latitudeDouble;
#endif

    // 6. Convertir les degres en QString
    latitude = QString::number(latitudeDouble);

#ifdef DEBUG_GPS
    //qDebug() << Q_FUNC_INFO << __LINE__ << latitude;
#endif

    // Pour la longitude
    // 1. Extraire dans le format dddmm.mmmm -> ddd
    longitudeD.remove(3, 10);

#ifdef DEBUG_GPS
    //qDebug() << Q_FUNC_INFO << __LINE__ << longitudeD;
#endif

    // 2. Extraire dans le format ddmm.mmmm -> mm.mmmm
    longitudeM.remove(0, 3);

#ifdef DEBUG_GPS
    //qDebug() << Q_FUNC_INFO << __LINE__ << longitudeM;
#endif

    // 3. Convertir ddd -> double
    longitudeDDouble = longitudeD.toDouble(&ok);

#ifdef DEBUG_GPS
    //qDebug() << Q_FUNC_INFO << __LINE__ << longitudeDDouble;
#endif

    // 4. Convertir mm.mmmm -> double
    longitudeMDouble = longitudeM.toDouble(&ok);

#ifdef DEBUG_GPS
    //qDebug() << Q_FUNC_INFO << __LINE__ << longitudeMDouble;
#endif

    // 5. Faire : 00448.776416 4 + (48.776416/60.0) = 4.812940267
    longitudeDouble = longitudeDDouble + (longitudeMDouble/60.0);

#ifdef DEBUG_GPS
    //qDebug() << Q_FUNC_INFO << __LINE__ << longitudeDouble;
#endif

    // 6. Convertir les degres en QString
    longitude = QString::number(longitudeDouble);

#ifdef DEBUG_GPS
    //qDebug() << Q_FUNC_INFO << __LINE__ << longitude;
#endif

#ifdef DEBUG_GPS
    qDebug() << Q_FUNC_INFO << __LINE__ << latitude << longitude;
#endif

    // 7. mise à jour de la localisation :
#ifndef SIMULATION_ET2
    if(latitude.isEmpty() || longitude.isEmpty())
        return;
    latitudeBus = latitude;
    longitudeBus = longitude;
#endif
}
```

9.2.3.8 GPS : :transmettre (const QString & *message*) [private]

Paramètres

message	

Renvoie

int

Références Port : :estOuvert(), port, et Port : :transmettre().

Référencé par demarrerGPS(), eteindreGPS(), et lire().

```
{
    QString delimiteur = "\r\n";

    if (!port->estOuvert())
    {
#ifdef DEBUG_GPS
        qDebug() << Q_FUNC_INFO << __LINE__ << QString::fromUtf8("port fermé");
#endif
        return 0;
    }

#ifdef DEBUG_GPS
        qDebug() << Q_FUNC_INFO << __LINE__ << message.toLatin1();
#endif
    int i = port->transmettre(message.toLatin1() + delimiteur.toLatin1());

    return i;
}
```

9.2.4 Documentation des données membres

9.2.4.1 bool GPS : :etatDemarre [private]

Un bool qui permet de vérifier l'état du gps (demarré/éteind)

Référencé par demarrerGPS(), eteindreGPS(), et lire().

9.2.4.2 QString GPS : :latitudeBus [private]

Un QString qui contient la latitude du bus

Référencé par getLatitude(), et traiterInfosGPS().

9.2.4.3 QString GPS : :longitudeBus [private]

Un QString qui contient la longitude du bus

Référencé par getLongitude(), et traiterInfosGPS().

9.2.4.4 Port* GPS : :port [private]

Pointeur sur un objet Port

Référencé par GPS(), recevoir(), transmettre(), et ~GPS().

La documentation de cette classe a été générée à partir des fichiers suivants :
– gps.h
– gps.cpp

9.3 Référence de la classe HautParleur

Classe qui permet de générer des messages sonores.

`#include <hautparleur.h>`

Fonctions membres publiques

- HautParleur ()
  Constructeur par défaut.
- ~HautParleur ()
  Destructeur.
- void liste ()
- void version ()
- void parle (QString text)
- int getVitesse () const
- void setVitesse (int vitesse)
- int getVolume () const
- void setVolume (int volume)
- int getTonalite () const
- void setTonalite (int tonalite)
- int getGamme () const
- void setGamme (int gamme)
- int getPonctuation () const
- void setPonctuation (int ponctuation)
- int getGenre () const
- void setGenre (int genre)
- int getTauxEchantillonnage () const
- void setTauxEchantillonnage (int tauxEchantillonnage)
- string getVoix () const
- void setVoix (string voix)

Attributs privés

- string voix
- int tauxEchantillonnage
- espeak_VOICE * voix_spec

9.3.1 Documentation des constructeurs et destructeur

9.3.1.1 HautParleur : :HautParleur ( )

```
{
        voix_spec = espeak_GetCurrentVoice();
        this->setVitesse(80);
        this->setVolume(80);
        this->setTonalite(50);
        this->setGamme(60);
        this->setPonctuation(0);
        this->setGenre(2);
        this->tauxEchantillonnage = espeak_Initialize(AUDIO_OUTPUT_PLAYBACK, 0,
      NULL, 0);
        voix = "mb-fr1"; // ("fr") ou ("fr+f3")
        this->setVoix(voix);
    qDebug() << "Taux Echantillonnage " << tauxEchantillonnage ;
}
```

9.3.1.2 HautParleur : :~HautParleur ( )

```
{
}
```

9.3.2 Documentation des fonctions membres

9.3.2.1 HautParleur : :getGamme ( ) const

Renvoie

int

```
{

        return espeak_GetParameter(espeakRANGE, 1);

}
```

9.3.2.2 HautParleur : :getGenre ( ) const

Renvoie

int

```
{
        return voix_spec->gender;
}
```

9.3.2.3 HautParleur : :getPonctuation ( ) const

Renvoie

int

```
{
        return espeak_GetParameter(espeakPUNCTUATION, 1);
}
```

9.3.2.4 HautParleur : :getTauxEchantillonnage ( ) const

Renvoie

int

```
{
        return this->tauxEchantillonnage;
}
```

9.3.2.5 HautParleur : :getTonalite ( ) const

Renvoie

int

```
{
        return espeak_GetParameter(espeakPITCH, 1);
}
```

9.3.2.6 HautParleur : :getVitesse ( ) const

Renvoie

int

```
{
        return espeak_GetParameter(espeakRATE, 1);
}
```

9.3.2.7 HautParleur : :getVoix ( ) const

Renvoie

string

```
{
    return this->voix;
}
```

9.3.2.8 HautParleur : :getVolume ( ) const

Renvoie

int

```
{

        return espeak_GetParameter(espeakVOLUME, 1);

}
```

9.3.2.9 HautParleur : :liste ( )

```
{
   const espeak_VOICE **voices = NULL;
   espeak_VOICE voice_select;
   const espeak_VOICE *v;

   voices = espeak_ListVoices(NULL);

   for(int ix=0; (v = voices[ix]) != NULL; ix++)
   {
      qDebug() << "Language shortsign : " << v->languages << endl;
      qDebug() << "Identifier : " << v->identifier << endl;
      qDebug() << "Age : " << v->age << endl;
      qDebug() << "Genre : " << v->gender << endl;
      qDebug() << "Nom: " << v->name << endl;
      qDebug() << "Variante : " << v->variant << endl;
      qDebug() << endl;
   }
}
```

9.3.2.10 HautParleur : :parle (QString *text*)

Paramètres

text	

Référencé par TInformationVoyageur : :signalerArret().

```
{
    qDebug() << "-- " << text << endl;
    espeak_Synth((char *)text.toAscii().data(), text.size(), 0, POS_CHARACTER,
      0, espeakCHARS_AUTO, NULL, NULL);
    espeak_Synchronize();
}
```

9.3.2.11 HautParleur : :setGamme (int *gamme*)

Paramètres

gamme	

```
{

     qDebug() << "Gamme : " << gamme << endl;

    if(gamme >= 0 && gamme <= 100)
    {
      espeak_SetParameter(espeakRANGE, gamme, 0);
    }

}
```

9.3.2.12 HautParleur : :setGenre (int *genre*)

Paramètres

genre	

```
{

        voix_spec->gender = genre;

}
```

9.3.2.13 HautParleur : :setPonctuation (int *ponctuation*)

Paramètres

ponctuation	

```
{

    qDebug() << "Ponctuation : " << ponctuation << endl;

    if(ponctuation >= 0 && ponctuation <= 1)
    {
      espeak_SetParameter(espeakPUNCTUATION, ponctuation, 0);
    }

}
```

9.3.2.14 HautParleur : :setTauxEchantillonnage (int *tauxEchantillonnage*)

Paramètres

tauxEchantillonnage	

9.3.2.15 HautParleur : :setTonalite (int *tonalite*)

Paramètres

tonalite	

```
{

    qDebug() << "Tonalite : " << tonalite << endl;

    if(tonalite >= 0 && tonalite <= 100)
    {
      espeak_SetParameter(espeakPITCH, tonalite, 0);
    }

}
```

9.3.2.16 HautParleur : :setVitesse (int *vitesse*)

Paramètres

vitesse	

```
{

    qDebug() << "Vitesse : " << vitesse << endl;
    if(vitesse >= 80 && vitesse <= 450)
    {
      int statut=espeak_SetParameter(espeakRATE, vitesse, 0);
      if (statut== EE_OK)
      qDebug()<<"OK";
    }

}
```

9.3.2.17 HautParleur : :setVoix (string *voix*)

Paramètres

voix	

```
{

        this->voix = voix;
        if(espeak_SetVoiceByName(this->voix.c_str()) != EE_OK)
        cerr << "espeak_SetVoiceByName error !" << endl;
    else
    {
        qDebug() << "Selection language : " << 'voix' << endl;
    }

}
```

9.3.2.18 HautParleur : :setVolume (int *volume*)

Paramètres

volume	

```
{

     qDebug() << "Volume : " << volume << endl;

    if(volume >= 0 && volume <= 100)
    {
      espeak_SetParameter(espeakVOLUME, volume, 0);
    }

}
```

9.3.2.19 HautParleur : :version ( )

```
{
    const char *eSpeakVersionInfo = NULL;

    eSpeakVersionInfo = espeak_Info(NULL);
    qDebug() << "HautParleur version : " << eSpeakVersionInfo << endl;
}
```

9.3.3 Documentation des données membres

9.3.3.1 int HautParleur : :tauxEchantillonnage [private]

Plus la fréquence est grande, plus la qualité du signal est élevée.

9.3.3.2 string HautParleur : :voix [private]

Voix à utiliser pour la synthese vocale.

9.3.3.3 espeak_VOICE* HautParleur : :voix_spec [private]

La documentation de cette classe a été générée à partir des fichiers suivants :
– hautparleur.h
– hautparleur.cpp

9.4 Référence de la structure InformationsConducteur

`#include <tgestionconducteur.h>`

Attributs publics

– QString idItineraire
– QString numLigne
– QString destination
– QString prochainArret
– QString heureDepart
– QString numArret
– int avanceRetard

9.4.1 Documentation des données membres

9.4.1.1 int InformationsConducteur : :avanceRetard

Référencé par TGestionConducteur : :TGestionConducteur().

9.4.1.2 QString InformationsConducteur : :destination

Référencé par TGestionConducteur : :actualiserInformationsVoyageur(), TGestionConducteur : :main(), et TGestionConducteur-: :TGestionConducteur().

9.4.1.3 QString InformationsConducteur : :heureDepart

Référencé par TGestionConducteur : :TGestionConducteur().

9.4.1.4 QString InformationsConducteur : :idItineraire

Référencé par TGestionConducteur : :actualiserInformationsVoyageur(), TGestionConducteur : :demarrerCourse(), TGestionConducteur : :main(), et TGestionConducteur : :TGestionConducteur().

9.4.1.5 QString InformationsConducteur : :numArret

Référencé par TGestionConducteur : :actualiserInformationsVoyageur(), TGestionConducteur : :main(), et TGestionConducteur : :TGestionConducteur().

9.4.1.6 QString InformationsConducteur : :numLigne

Référencé par TGestionConducteur : :actualiserInformationsVoyageur(), TGestionConducteur : :main(), et TGestionConducteur : :TGestionConducteur().

9.4.1.7 QString InformationsConducteur : :prochainArret

Référencé par TGestionConducteur : :actualiserInformationsVoyageur(), TGestionConducteur : :main(), et TGestionConducteur : :TGestionConducteur().

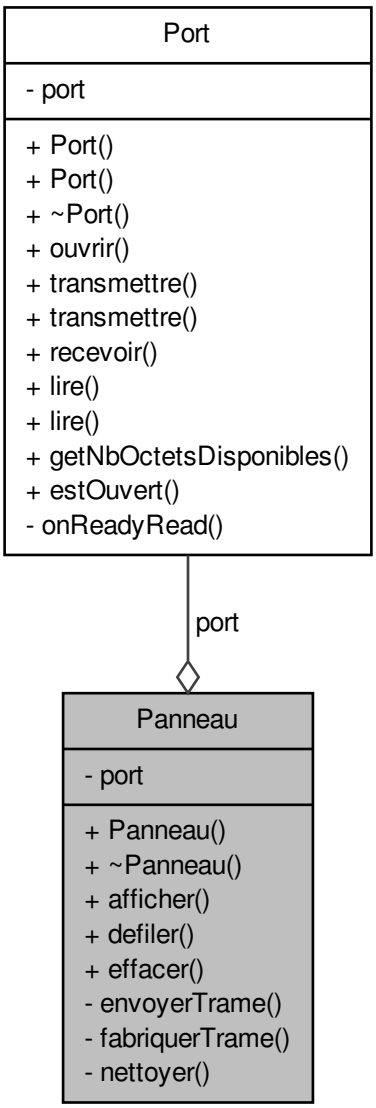
La documentation de cette structure a été générée à partir du fichier suivant :
– tgestionconducteur.h

9.5 Référence de la classe Panneau

Classe qui gère l'affichage d'informations sur le panneau.

`#include <panneau.h>`

Graphe de collaboration de Panneau :



Fonctions membres publiques

- Panneau (QString portPanneau=PORT_DEFAUT)
  Constructeur de la classe Panneau.
- ~Panneau ()
  Destructeur de la classe Panneau.
- void afficher (QString message, int duree, int position=POS_DEFAUT)
  Affichage fixe d'un message sur le Panneau (< 10 caractères)
- void defiler (QString message, int vitesse, int position=POS_DEFAUT)
  Défilement de texte sur le Panneau (> 10 caractères)
- void effacer (int duree)
  Permet d'effacer le Panneau pour un prochain affichage.

Fonctions membres privées

- bool envoyerTrame (char trame[MAX_TRAME])
  Envoie la trame sur le panneau.

– int fabriquerTrame (char *trame, QString message, int position=POS_DEFAUT)
  Fabrique la trame en fonction du protocole.
– QString nettoyer (QString message)

Attributs privés

– Port * port

9.5.1 Description détaillée

Auteur

Gabriel Destrée <gabriel.destree@gmail.com>

9.5.2 Documentation des constructeurs et destructeur

9.5.2.1 Panneau : :Panneau (QString *portPanneau* = PORT_DEFAUT)

Paramètres

portPanneau	QString contenant le chemin vers le fichier de périphérique (valeur par défaut "/dev/panneau")

```
{
    #ifdef DEBUG_PANNEAU
    qDebug() << Q_FUNC_INFO << "-> port : " << portPanneau;
    #endif

    #ifndef SIMULATION_PANNEAU
    port = new Port(portPanneau);
    #else
    Q_UNUSED(port)
    #endif
}
```

9.5.2.2 Panneau : :~Panneau ( )

```
{
    #ifdef DEBUG_PANNEAU
    qDebug() << Q_FUNC_INFO;
    #endif
    #ifndef SIMULATION_PANNEAU
    delete port;
    #endif
}
```

9.5.3 Documentation des fonctions membres

9.5.3.1 Panneau : :afficher (QString *message,* int *duree,* int *position* = POS_DEFAUT)

Paramètres

message	QString contenant la chaîne de caractère à afficher (valeur par défaut : aucune)
duree	entier contenant la durée de maintien de l'affichage exprimée en secondes (valeur par défaut : aucune)
position	entier contenant la position sur l'afficheur du premier caractère du message (valeur par défaut 11)

Références FIN, et MAX_TRAME.

Référencé par TDiffusionMessages : :afficherArret(), TDiffusionMessages : :afficherDate(), TDiffusionMessages : :afficherDestination(), TDiffusionMessages : :afficherHeure(), et TDiffusionMessages : :afficherLigne().

```
{
    char trame[MAX_TRAME] = {FIN};

    #ifdef DEBUG_PANNEAU
    qDebug() << Q_FUNC_INFO << "-> message : " << message << "position : " <<
      position;
    #endif

    // on fabrique la trame
    fabriquerTrame(&trame[0], message, position);

    // on envoie la trame
```

```
    envoyerTrame(trame);

    sleep(duree);
}
```

9.5.3.2 Panneau : :defiler (QString *message*, int *vitesse*, int *position* = POS_DEFAUT)

Paramètres

message	QString contenant la chaîne de caractère à afficher (valeur par défaut : aucune)
vitesse	entier contenant la vitesse de défilement exprimée en millisecondes (valeur par défaut : aucune)
position	entier contenant la position sur l'afficheur du premier caractère du message (valeur par défaut 11)

Références FIN, MAX_TEXTE, et MAX_TRAME.

Référencé par TDiffusionMessages : :afficherArret(), TDiffusionMessages : :afficherDestination(), et TDiffusionMessages-: :afficherLigne().

```
{
    char trame[MAX_TRAME] = {FIN};
    QString texte;

    #ifdef DEBUG_PANNEAU
    qDebug() << Q_FUNC_INFO << "-> message : " << message << "position : " <<
      position;
    #endif
    for (int i = 0 ; i < message.length() ; i++)
    {
        texte = message.mid(i, MAX_TEXTE);
#ifdef DEBUG_PANNEAU
//qDebug() << Q_FUNC_INFO << "-> texte : " << texte;
#endif
        fabriquerTrame(&trame[0], texte, position);
        envoyerTrame(trame);

        usleep(vitesse*1000);
    }
}
```

9.5.3.3 Panneau : :effacer (int *duree*)

Paramètres

duree	entier contenant la durée de maintien de l'affichage exprimée en secondes (valeur par défaut : aucune)

Références FIN, et MAX_TRAME.

Référencé par TDiffusionMessages : :diffuser().

```
{
    char trame[MAX_TRAME] = {FIN};
    QString message = "";

    #ifdef DEBUG_PANNEAU
    qDebug() << Q_FUNC_INFO;
    #endif

    // on fabrique la trame
    fabriquerTrame(&trame[0], message);

    // on envoie la trame
    envoyerTrame(trame);

    sleep(duree);
}
```

9.5.3.4 Panneau : :envoyerTrame (char *trame[MAX_TRAME]*) [private]

Paramètres

trame[]	un tableau de MAX_TRAME caractères

Renvoie

bool vrai si la trame a été envoyée entièrement sinon faux

```
{
```

```
    int retour = 0;
    bool etat = false;

    #ifndef SIMULATION_PANNEAU
    retour = port->transmettre(trame);
    #else
    Q_UNUSED(trame)
    retour = 1; // ok
    #endif

    // si l'envoi de la trame echoue ?
    if (retour < 1)
    {
        #ifdef DEBUG_PANNEAU
        qDebug() << Q_FUNC_INFO << "-> erreur emission trame !";
        #endif
        etat = false;
    }
    else
    {
        if(retour == (int)strlen(trame))
            etat = true;
        else
            etat = false;
    }

    return etat;
}
```

9.5.3.5 Panneau : :fabriquerTrame (char * *trame*, QString *message*, int *position* = POS_DEFAUT) [private]

Paramètres

trame	pointeur sur un tableau de MAX_TRAME caractères
message	QString contenant le message à afficher
position	entier contenant la position sur l'afficheur du premier caractère du message (valeur par défaut 11)

Renvoie

int entier indiquant si la trame a été fabriquée

Références AUX, CHAN, CMD, EOT, ETX, F, FIN, LINE, MAX_TRAME, NUM, SOT, STX, et Z.

```
{
    char trameposition[4]= {FIN};
    QString messageNettoye;

    // réinitialise la trame à 0
    memset(&trame[0], 0, MAX_TRAME);

    trame[0] = SOT;
    trame[1] = NUM;
    trame[2] = NUM;
    trame[3] = CMD;
    trame[4] = AUX;
    trame[5] = F;
    trame[6] = F;
    trame[7] = AUX;
    trame[8] = Z;
    trame[9] = Z;
    trame[10] = STX;
    trame[11] = CHAN;
    trame[12] = LINE;

    sprintf(trameposition,"%02d",position);

    trame[13] = trameposition[0];
    trame[14] = trameposition[1];

    int i = 15;
    int j = 0;

    messageNettoye = nettoyer(message);

    // 17 octets de protocole (taille fixe)
    for (j = 0 ; j < messageNettoye.length() && j < (MAX_TRAME - 17) ; j++)
    {
        trame[i++] = messageNettoye.at(j).toAscii();
    }

    trame[i] = ETX;
    trame[i+1] = EOT;

    return 1;
}
```

9.5.3.6 QString Panneau : :nettoyer (QString *message*) [private]

```
{
    QString messageNettoye;

    messageNettoye = message.replace(QString::fromUtf8("ç"), "c");
    messageNettoye = message.replace(QString::fromUtf8("â"), "a");
    messageNettoye = message.replace(QString::fromUtf8("ê"), "e");
    messageNettoye = message.replace(QString::fromUtf8("î"), "i");
    messageNettoye = message.replace(QString::fromUtf8("ô"), "o");
    messageNettoye = message.replace(QString::fromUtf8("û"), "u");
    messageNettoye = message.replace(QString::fromUtf8("à"), "a");
    messageNettoye = message.replace(QString::fromUtf8("è"), "e");
    messageNettoye = message.replace(QString::fromUtf8("é"), "e");
    messageNettoye = message.replace(QString::fromUtf8("ï"), "i");
    messageNettoye = message.replace(QString::fromUtf8("ë"), "e");

    return messageNettoye;
}
```

9.5.4 Documentation des données membres

9.5.4.1 Port* Panneau : :port [private]

Pointeur sur un objet Port

La documentation de cette classe a été générée à partir des fichiers suivants :
– panneau.h
– panneau.cpp

9.6 Référence de la classe Port

Classe de gestion d'un port série.

`#include <port.h>`

Fonctions membres publiques

– Port ()
  Constructeur par défaut.
– Port (const QString &portName)
  Constructeur.
– ~Port ()
  Destructeur.
– bool ouvrir (const QString &portName, int debit=9600)
  Ouvre et configure un port série.
– int transmettre (const QString &message)
  Transmet un message sur le port série.
– int transmettre (char *message)
  Transmet un message sur le port série.
– int recevoir (QString &message)
  Réceptionne un message sur le port série.
– char lire ()
  Lit un caractère sur le port série.
– int lire (char *buffer, int n)
  Lit n caractères sur le port série.
– long getNbOctetsDisponibles ()
  Retourne le nombre d'octets disponible pour une lecture.
– bool estOuvert ()
  Retourne l'état du port série.

Connecteurs privés

– void onReadyRead ()
  Slot non utilisé

Attributs privés

– QextSerialPort * port

9.6.1 Description détaillée

Auteur

Thierry Vaira <tvaira@free.fr>

9.6.2 Documentation des constructeurs et destructeur

9.6.2.1 Port : :Port ( )

```
          :port(NULL)
{
}
```

9.6.2.2 Port : :Port (const QString & *portName*)

Paramètres

portName	QString indiquant le nom et chemin vers le fichier de périphérique

Références ouvrir().

```
                                  :port(NULL)
{
#ifdef DEBUG_PORT

   qDebug() << "<Port::Port()> port : " << portName;
#endif


   if(!portName.isEmpty())
   {
       ouvrir(portName, 9600);
   }
}
```

9.6.2.3 Port : :~Port ( )

Références port.

```
{
   if (port != NULL && port->isOpen())
      port->close();
}
```

9.6.3 Documentation des fonctions membres

9.6.3.1 Port : :estOuvert ( )

Renvoie

bool vrai si le port est ouvert sinon faux

Références port.

Référencé par GPS : :recevoir(), et GPS : :transmettre().

```
{
    if (port != NULL)
        return port->isOpen();
    else
        return false;
}
```

9.6.3.2 Port : :getNbOctetsDisponibles ( )

Renvoie

long nombre d'octets disponible pour une lecture

Références port.

Référencé par GPS : :recevoir().

```
{
   return port->bytesAvailable();
}
```

9.6.3.3 Port : :lire ( )

Renvoie

char le caractère reçu

Références port.

Référencé par GPS : :recevoir().

```
{
   if (port == NULL || !port->isOpen())
      return -1;

   char c;
   int n = port->read(&c, 1);
   if (n == -1)
       c = '\0'; // fin de chaine

   //printf("%c (0x%02X)\n", c, c);

   return c;
}
```

9.6.3.4 Port : :lire (char * *buffer*, int *n*)

Paramètres

buffer	pointeur sur un tableau de caractères
n	nombre de caractères à lire

Renvoie

int nombre de caractères lus

Références port.

```
{
    if (port == NULL || !port->isOpen())
       return -1;

    return port->read(buffer, n);
}
```

9.6.3.5 Port : :onReadyRead ( ) [private, slot]

Références port.

Référencé par ouvrir().

```
{
   QByteArray datas;

   int a = port->bytesAvailable();
   datas.resize(a);

   port->read(datas.data(), datas.size());
   #ifdef DEBUG_PORT

   qDebug() << "<Port::onReadyRead()> nb octets lus : " << datas.size();
   qDebug() << "<Port::onReadyRead()> donnees recues :" << datas;
   #endif


   #ifdef DEBUG_PORT
```

```
    for(int i=0;i<datas.size();i++)
       printf("0x%02X ", (unsigned char)datas.data()[i]);
    printf("\n\n");
    #endif

}
```

9.6.3.6 Port : :ouvrir (const QString & *portName*, int *debit* = 9600)

Paramètres

portName	QString indiquant le nom et chemin vers le fichier de périphérique
debit	entier précisant le débit (9600 bits/s par défaut)

Renvoie

bool vrai si le port est ouvert sinon faux

Références onReadyRead(), et port.

Référencé par Port().

```
{
    if (port != NULL && port->isOpen())
       port->close();

    // mode synchrone : QextSerialPort::EventDriven (Can only be used with
        threads started with QThread)
    //port = new QextSerialPort(QLatin1String("/dev/ttyAMA0"),
        QextSerialPort::EventDriven);
    // ou
    // mode asynchrone : QextSerialPort::Polling
    this->port = new QextSerialPort(portName, QextSerialPort::Polling);
    switch(debit)
    {
    case 9600 :
        port->setBaudRate(BAUD9600);
        break;
    case 4800 :
        port->setBaudRate(BAUD4800);
        break;
    }
    port->setFlowControl(FLOW_OFF);
    port->setParity(PAR_NONE);
    port->setDataBits(DATA_8);
    port->setStopBits(STOP_1);
    port->setTimeout(500);

    //port->open(QIODevice::ReadWrite | QIODevice::Unbuffered);
    if (port->open(QIODevice::ReadWrite) == true)
    {
       connect(port, SIGNAL(readyRead()), this, SLOT(onReadyRead()));
       #ifdef DEBUG_PORT

       qDebug() << "<Port::ouvrir()> port " << port->portName() << " ouvert";
       #endif

    }
    else
    {
       #ifdef DEBUG_PORT

       qDebug() << "<Port::ouvrir()> Erreur ouverture port " << port->portName()
       ;
       #endif

    }

    return port->isOpen();
}
```

9.6.3.7 Port : :recevoir (QString & *message*)

Paramètres

message	le message reçu

Renvoie

int nombre d'octets reçus

Références BUFFERSIZE, et port.

```
{
   char buffer[BUFFERSIZE+1]; // + fin de chaine

   int nbOctets = -1;

   if (port == NULL || !port->isOpen())
      return -1;

   nbOctets = port->bytesAvailable();
   if(nbOctets > BUFFERSIZE)
       nbOctets = BUFFERSIZE;
   #ifdef DEBUG_PORT

   if(nbOctets != 0)
      qDebug() << "<Port::recevoir()> octets recus disponibles : %d" <<
      nbOctets;
   #endif


   int n = port->read(buffer, nbOctets);
   if (n != -1)
       buffer[n] = '\0'; // fin de chaine
   else
       buffer[0] = '\0'; // fin de chaine

   message = QLatin1String(buffer);

   #ifdef DEBUG_PORT

   if(n != 0)
   {
      qDebug() << "<Port::recevoir()> octets lus : %d" << n;
      for(int i=0;i<n;i++)
         printf("0x%02X ", (unsigned char)buffer[i]);
      printf("\n\n");
   }
   #endif


   return n;
}
```

9.6.3.8 Port : :transmettre (const QString & *message*)

Paramètres

message	const QString & le message à transmettre

Renvoie

int nombre de caractères transmis

Références port.

Référencé par GPS : :transmettre().

```
{
   int nbOctets = -1;

   if (port == NULL || !port->isOpen())
      return -1;

   nbOctets = port->write(message.toLatin1());

   #ifdef DEBUG_PORT

   qDebug("<Port::transmettre()> octets transmis : %d", nbOctets);
   qDebug() << "<Port::transmettre()> message transmis : " << message;
   #endif


   return nbOctets;
}
```

9.6.3.9 Port : :transmettre (char * *message*)

Paramètres

message	char * pointeur sur le message à transmettre

Renvoie

int nombre de caractères transmis

Références port.

```
{
   int nbOctets = -1;

   if (port == NULL || !port->isOpen())
      return -1;

   nbOctets = port->write(message);

   #ifdef DEBUG_PORT

   qDebug("<Port::transmettre()> octets transmis : %d", nbOctets);
   #endif


   return nbOctets;
}
```

9.6.4 Documentation des données membres

9.6.4.1 QextSerialPort* Port : :port [private]

pointeur sur un objet QextSerialPort

Référencé par estOuvert(), getNbOctetsDisponibles(), lire(), onReadyRead(), ouvrir(), recevoir(), transmettre(), et ~Port().

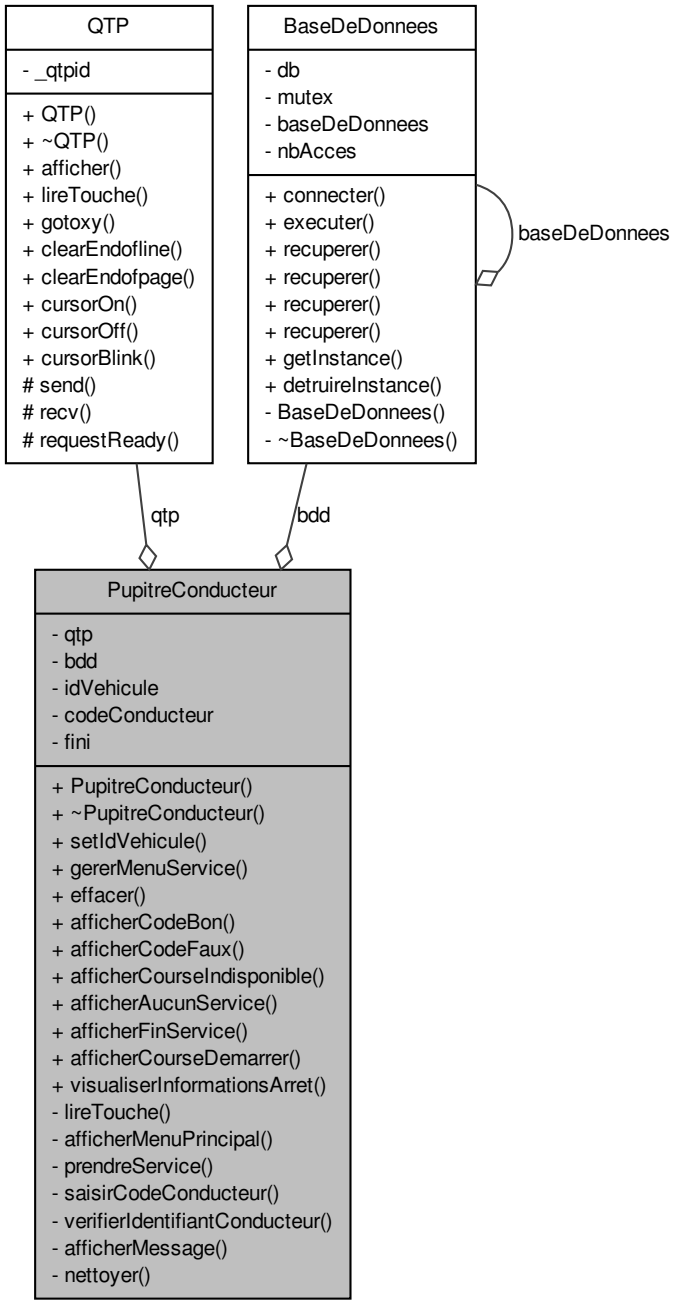
La documentation de cette classe a été générée à partir des fichiers suivants :
– port.h
– port.cpp

9.7 Référence de la classe PupitreConducteur

Classe qui gère le pupitre du conducteur.

`#include <pupitreconducteur.h>`

Graphe de collaboration de PupitreConducteur :



Fonctions membres publiques

- PupitreConducteur (QString port="/dev/qtp")
  Constructeur de la classe PupitreConducteur.
- ~PupitreConducteur ()
  Destructeur de la classe PupitreConducteur.
- void setIdVehicule (QString idVehicule)
  Permet de déterminer le numéro du véhicule.
- int gererMenuService (QString &donnee, int etatService, int enCourse=0)
  Gère le menu principal du pupitre.
- void effacer ()
  Efface l'écran du pupitre.

- void afficherCodeBon ()
  Affiche que le code saisi est bon.
- void afficherCodeFaux ()
  Affiche que le code saisi est faux.
- void afficherCourseIndisponible ()
  Affiche quand il n'y a pas/plus de course à effectuée.
- void afficherAucunService ()
  Affiche quand il n'y a pas de service actif.
- void afficherFinService ()
  Affiche quand on met fin au service.
- void afficherCourseDemarrer ()
  Affiche quand la course est demarrer.
- bool visualiserInformationsArret (QString idItineraire, QString numLigne, QString destination, QString prochainArret, QString numArret)
  Affiche les informations d'une courses et arret.

Fonctions membres privées

- char lireTouche (char *touche, int nbchar, int mode)
  Permet de lire une touche saisis.
- void afficherMenuPrincipal ()
  Affiche le menu principal du pupitre.
- bool prendreService ()
  Affiche le menu service et permet de prendre son service.
- void saisirCodeConducteur ()
  Permet la saisie du code conducteur.
- bool verifierIdentifiantConducteur (QString code)
  Verifie si le code conducteur est correct avec la base de données.
- void afficherMessage (char message[][LG_LIGNE+1])
  Affiche un message sur le pupitre.
- QString nettoyer (QString message)
  permet de nettoyé un message pour les codes ASCII

Attributs privés

- QTP * qtp
- BaseDeDonnees * bdd
- QString idVehicule
- QString codeConducteur
- bool fini

9.7.1 Description détaillée

Auteur

Cyril Cambe <cambecyril@yahoo.fr>

9.7.2 Documentation des constructeurs et destructeur

9.7.2.1 PupitreConducteur : :PupitreConducteur (QString *port* = "/dev/qtp")

Paramètres

port	

Références bdd, BaseDeDonnees : :getInstance(), et qtp.

```
                                                    : qtp(NULL), bdd(NULL),
    idVehicule(""), codeConducteur(""), fini(false)
{
    #ifdef DEBUG_QTP
    qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
    #endif
    #ifndef SIMULATION_PUPITRE
    qtp = new QTP(port);
    #else
    Q_UNUSED(port)
    #endif

    bdd = BaseDeDonnees::getInstance();
}
```

9.7.2.2 PupitreConducteur : :~PupitreConducteur ( )

Références qtp.

```
{
    #ifdef DEBUG_QTP
    qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
    #endif
    #ifndef SIMULATION_PUPITRE
    delete qtp;
    #endif
}
```

9.7.3 Documentation des fonctions membres

9.7.3.1 PupitreConducteur : :afficherAucunService ( )

Références afficherMessage(), LG_LIGNE, et NB_LIGNES.

Référencé par gererMenuService().

```
{
    char messages[NB_LIGNES][LG_LIGNE+1] =
    {
         {"*                   *"}
         ,{"      SERVICE        "}
         ,{"     NON ACTIF       "}
         ,{"                     "}
    };
    afficherMessage(messages);
    sleep(1);
}
```

9.7.3.2 PupitreConducteur : :afficherCodeBon ( )

Références afficherMessage(), LG_LIGNE, et NB_LIGNES.

Référencé par prendreService().

```
{
    char messages[NB_LIGNES][LG_LIGNE+1] =
    {
         {"*                   *"}
         ,{"      SERVICE        "}
         ,{"        OK           "}
         ,{"                     "}
    };
    afficherMessage(messages);
    sleep(1);
}
```

9.7.3.3 PupitreConducteur : :afficherCodeFaux ( )

Références afficherMessage(), LG_LIGNE, et NB_LIGNES.

Référencé par prendreService().

```
{
    char messages[NB_LIGNES][LG_LIGNE+1] =
    {
         {"*                   *"}
         ,{"       AUCUN         "}
         ,{"      SERVICE        "}
         ,{"                     "}
    };
    afficherMessage(messages);
    sleep(1);
}
```

9.7.3.4 PupitreConducteur : :afficherCourseDemarrer ( )

Références afficherMessage(), LG_LIGNE, et NB_LIGNES.

Référencé par TGestionConducteur : :main().

```
{
    char messages[NB_LIGNES][LG_LIGNE+1] =
    {
         {"*                   *"}
         ,{"      COURSE         "}
         ,{"     DEMARRER        "}
         ,{"                     "}
    };
    afficherMessage(messages);
    sleep(1);
}
```

9.7.3.5 PupitreConducteur : :afficherCourseIndisponible ( )

Références afficherMessage(), LG_LIGNE, et NB_LIGNES.

Référencé par TGestionConducteur : :main().

```
{
    char messages[NB_LIGNES][LG_LIGNE+1] =
    {
         {"*                   *"}
         ,{"   AUCUNE COURSE    "}
         ,{"     DISPONIBLE     "}
         ,{"                     "}
    };
    afficherMessage(messages);
    sleep(1);
}
```

9.7.3.6 PupitreConducteur : :afficherFinService ( )

Références afficherMessage(), LG_LIGNE, et NB_LIGNES.

Référencé par TGestionConducteur : :terminerService().

```
{
    char messages[NB_LIGNES][LG_LIGNE+1] =
    {
         {"*                   *"}
         ,{"        FIN          "}
         ,{"      SERVICE        "}
         ,{"                     "}
    };
    afficherMessage(messages);
    sleep(1);
}
```

9.7.3.7 PupitreConducteur : :afficherMenuPrincipal ( ) [private]

Références afficherMessage(), LG_LIGNE, et NB_LIGNES.

Référencé par gererMenuService().

```
{
    char messages[NB_LIGNES][LG_LIGNE+1] =
    {
        {"1. PRISE DE SERVICE"}
        ,{"2. FIN DE SERVICE"}
        ,{"3. DEMARRER COURSE"}
        ,{"4. AFFICHER COURSE"}
    };

    afficherMessage(messages);
}
```

9.7.3.8 PupitreConducteur : :afficherMessage (char *message[][LG_LIGNE+1]*) [private]

Paramètres

message[][]	

Références QTP : :afficher(), QTP : :clearEndofline(), QTP : :cursorOff(), QTP : :gotoxy(), NB_LIGNES, et qtp.

Référencé par afficherAucunService(), afficherCodeBon(), afficherCodeFaux(), afficherCourseDemarrer(), afficherCourseIndisponible(), afficherFinService(), afficherMenuPrincipal(), saisirCodeConducteur(), et visualiserInformationsArret().

```
{
    int nbLignes = NB_LIGNES;
    int i;

    #ifdef SIMULATION_PUPITRE
    for(i=0;i<nbLignes;i++)
    {
        qDebug("%s", messages[i]);
    }
    #else
    qtp->cursorOff();
    for(i=0;i<nbLignes;i++)
    {
        qtp->gotoxy(i, 0);
        qtp->clearEndofline();
        qtp->afficher(messages[i], strlen(messages[i]));
    }
    #endif
}
```

9.7.3.9 PupitreConducteur : :effacer ( )

Références QTP : :clearEndofline(), QTP : :cursorOff(), QTP : :cursorOn(), QTP : :gotoxy(), NB_LIGNES, et qtp.

Référencé par TGestionConducteur : :main().

```
{
    int nbLignes = NB_LIGNES;
    int i;

    #ifdef SIMULATION_PUPITRE
    Q_UNUSED(nbLignes)
    Q_UNUSED(i)
    //system("clear");
    #else
    qtp->cursorOff();
    for(i=0;i<nbLignes;i++)
    {
        qtp->gotoxy(i, 0);
        qtp->clearEndofline();
    }
    qtp->gotoxy(0, 0);
    qtp->cursorOn();
    #endif
}
```

9.7.3.10 PupitreConducteur : :gererMenuService (QString & *donnee*, int *etatService*, int *enCourse* = 0)

Paramètres

donnee	
etatService	

Renvoie

int

Références AFFICHER_COURSE, afficherAucunService(), afficherMenuPrincipal(), AUCUN, codeConducteur, DEMARRER_COURSE, EN_SERVICE, FIN_SERVICE, fini, lireTouche(), QTP : :lireTouche(), PAS_EN_COURSE, PAS_EN_SERVICE, prendreService(), PRISE_SERVICE, qtp, et QTP_NOECHO.

Référencé par TGestionConducteur : :main().

```
{
    char touche;
    int choixService = AUCUN;
    bool etat;

    do
    {
        if(enCourse == PAS_EN_COURSE)
            afficherMenuPrincipal();

        #ifdef SIMULATION_PUPITRE
        lireTouche(&touche, 1, QTP_NOECHO);
```

```
        #else
        qtp->lireTouche(&touche, 1, QTP_NOECHO);
        #endif
        switch(touche)
        {
            case '1' :
                if(etatService == PAS_EN_SERVICE)
                {
                    codeConducteur = "";
                    etat = prendreService();
                    if(etat == true)
                    {
                        choixService = PRISE_SERVICE;
                        donnee = codeConducteur;
                    }
                    else
                    {
                        choixService = AUCUN;
                        donnee = "";
                    }
                    fini = true;
                }
                break;
            case '2' :
                if(etatService == EN_SERVICE)
                {
                    choixService = FIN_SERVICE;
                    fini = true;
                }
                else
                    afficherAucunService();
                break;
            case '3' :
                if(etatService == EN_SERVICE)
                {
                    choixService = DEMARRER_COURSE;
                    fini = true;
                }
                break;
            case '4' :
                if(etatService == EN_SERVICE)
                {
                    choixService = AFFICHER_COURSE;
                    fini = true;
                }
                break;
            /*case '5' :
                fini = true; // pour la démo
                break;*/
            default  :
                break;
        }
    }
    while(!fini);

    return choixService;
}
```

9.7.3.11 PupitreConducteur : :lireTouche (char * *touche*, int *nbchar*, int *mode*) [private]

Paramètres

touche	
nbchar	
mode	avec un echo sur l'afficheur ou pas

Renvoie

char

Références DIESE, et ETOILE.

Référencé par gererMenuService(), et saisirCodeConducteur().

```
{
    Q_UNUSED(mode)

    char key;
    int lus;

    for(lus=0;lus<nbchar;lus++)
    {
        cin >> key;
        *(touche+lus) = key;
    }
```

```
    // conversion de touche (key)
    if (key == 48) // 0x30
    {
        key = 0;
    }
    if (key == 49)
    {
        key = 1;
    }
    if (key == 50)
    {
        key = 2;
    }
    if (key == 51)
    {
        key = 3;
    }
    if (key == 52)
    {
        key = 4;
    }
    if (key == 53)
    {
        key = 5;
    }
    if (key == 54)
    {
        key = 6;
    }
    if (key == 55)
    {
        key = 7;
    }
    if (key == 56)
    {
        key = 8;
    }
    if (key == 57)
    {
        key = 9;
    }
    if (key == 0x23)
    {
        key = DIESE;
    }
    if (key == 0x2A)
    {
        key = ETOILE;
    }

    return key;
}
```

9.7.3.12 QString PupitreConducteur : :nettoyer (QString *message*) [private]

Paramètres

message	

Renvoie

QStrin

Référencé par visualiserInformationsArret().

```
{
    QString messageNettoye;

    messageNettoye = message.replace(QString::fromUtf8("ç"), "c");
    messageNettoye = message.replace(QString::fromUtf8("â"), "a");
    messageNettoye = message.replace(QString::fromUtf8("ê"), "e");
    messageNettoye = message.replace(QString::fromUtf8("î"), "i");
    messageNettoye = message.replace(QString::fromUtf8("ô"), "o");
    messageNettoye = message.replace(QString::fromUtf8("û"), "u");
    messageNettoye = message.replace(QString::fromUtf8("à"), "a");
    messageNettoye = message.replace(QString::fromUtf8("è"), "e");
    messageNettoye = message.replace(QString::fromUtf8("é"), "e");
    messageNettoye = message.replace(QString::fromUtf8("ï"), "i");
    messageNettoye = message.replace(QString::fromUtf8("ë"), "e");

    return messageNettoye;
}
```

9.7.3.13 PupitreConducteur : :prendreService ( ) [private]

Renvoie

bool

Références afficherCodeBon(), afficherCodeFaux(), codeConducteur, saisirCodeConducteur(), et verifierIdentifiantConducteur().

Référencé par gererMenuService().

```
{
    char touche;

    saisirCodeConducteur();

    if(!verifierIdentifiantConducteur(codeConducteur))
    {
        afficherCodeFaux();
        #ifdef DEBUG_QTP
        qDebug() << "Code faux";
        #endif
        return false;
    }
    else
    {
        afficherCodeBon();
        #ifdef DEBUG_QTP
        qDebug() << "Code bon";
        #endif
        return true;
    }
}
```

9.7.3.14 PupitreConducteur : :saisirCodeConducteur ( ) [private]

Références afficherMessage(), codeConducteur, CR, ESC, LG_LIGNE, lireTouche(), QTP : :lireTouche(), NB_LIGNES, qtp, et QTP_NOECHO.

Référencé par prendreService().

```
{
        char touche;

        char messages[NB_LIGNES][LG_LIGNE+1] =
        {
             {"*                    *"}
             ,{"  PRISE DE SERVICE  "}
             ,{"     SAISIR CODE    "}
             ,{"                    "}
        };

        afficherMessage(messages);

        do
        {
            #ifdef SIMULATION_PUPITRE
            lireTouche(&touche, 1, QTP_NOECHO);
            #else
            qtp->lireTouche(&touche, 1, QTP_NOECHO);
            #endif

            if(touche == ESC)
                break;
            else if(touche == CR)
                break;
            else
                codeConducteur += touche;
            #ifdef DEBUG_QTP
            qDebug() << codeConducteur ;
            #endif
        }
        while(touche != CR);
}
```

9.7.3.15 PupitreConducteur : :setIdVehicule (QString *idVehicule*)

Paramètres

idVehicule	

Références bdd, BaseDeDonnees : :connecter(), et idVehicule.

Référencé par TGestionConducteur : :TGestionConducteur().

```
{
    this->idVehicule = idVehicule;
    bdd->connecter("referentiel-" + idVehicule + ".sqlite");
}
```

9.7.3.16 PupitreConducteur : :verifierIdentifiantConducteur (QString *code*) [private]

Paramètres

code	

Renvoie

bool

Références bdd, et BaseDeDonnees : :recuperer().

Référencé par prendreService().

```
{
    QString code;
    QString requete = "SELECT codeConducteur FROM service WHERE
       codeConducteur='"+ codeConducteur + "' AND effectue=0";

    bool etat = bdd->recuperer(requete, code);

    if(etat == true)
    {
        if(code == codeConducteur)
            return true;
        else
            return false;
    }

    return false;
}
```

9.7.3.17 PupitreConducteur : :visualiserInformationsArret (QString *idItineraire,* QString *numLigne,* QString *destination,* QString *prochainArret,* QString *numArret*)

Références afficherMessage(), bdd, LG_LIGNE, NB_LIGNES, nettoyer(), et BaseDeDonnees : :recuperer().

Référencé par TGestionConducteur : :actualiserInformationsVoyageur(), et TGestionConducteur : :main().

```
{
    QStringList informationsArret;
    QString heureDepart = "";
    QString ligne1 = "", ligne2 = "", ligne3 = "", ligne4 = "";
    char messages[NB_LIGNES][LG_LIGNE+1] =
    {
         {"                    "}
        ,{"                    "}
        ,{"                    "}
        ,{"                    "}
    };
    QTime maintenant = QTime::currentTime();
    QString heure = maintenant.toString("hh:mm");

    #ifdef DEBUG_QTP
    qDebug() << Q_FUNC_INFO << numLigne << destination << prochainArret <<
      numArret;
    #endif

    if(numArret.length() > 0)
    {
        QString requete = "SELECT arret.heureDepart FROM arret WHERE
       idItineraire="+ idItineraire +" and numeroSequence="+numArret;
        bool etat = bdd->recuperer(requete, informationsArret);

        #ifdef DEBUG_QTP
        qDebug() << Q_FUNC_INFO << requete;
        #endif
        if(etat == true)
        {
            #ifdef DEBUG_QTP
            qDebug() << Q_FUNC_INFO << informationsArret << endl;
            #endif
            heureDepart = informationsArret.at(0);
        }
```

```
        else
        {
            #ifdef DEBUG_QTP
            qDebug() << Q_FUNC_INFO << "erreur requete !";
            #endif
            heureDepart = informationsArret.at(0);
        }
    }

    if(numLigne.length() > 0)
    {
        ligne1 = "LIGNE : " + numLigne + "     " + heure;
    }
    else
    {
        ligne1 = "En attente ...";
    }
    strcpy(messages[0], ligne1.toAscii().data());

    if(destination.length() < LG_LIGNE)
    {
        ligne2 = destination;
        ligne2 = nettoyer(ligne2);
    }
    else
    {
        // cf. mid()
        ligne2 = destination.mid(0, LG_LIGNE);
        ligne2 = nettoyer(ligne2);
    }
    strcpy(messages[1], ligne2.toAscii().data());

    if(prochainArret.length() < LG_LIGNE)
    {
        ligne3 = prochainArret;
        ligne3 = nettoyer(ligne3);
    }
    else
    {
        ligne3 = prochainArret.mid(0, LG_LIGNE);
        ligne3 = nettoyer(ligne3);
    }
    strcpy(messages[2], ligne3.toAscii().data());

    if(numLigne.length() > 0)
    {
        ligne4 = "Depart : " + heureDepart;
        strcpy(messages[3], ligne4.toAscii().data());
    }

    afficherMessage(messages);
    if(numLigne.length() == 0)
    {
        sleep(1);
    }

    //char touche = 0;
    //do
    /*{
        #ifdef SIMULATION_PUPITRE
        lireTouche(&touche, 1, QTP_NOECHO);
        #else
        qtp->lireTouche(&touche, 1, QTP_NOECHO);
        #endif
    }
    //while(touche != ESC);
    if(touche == ESC)
        return true;*/
    return false;
}
```

9.7.4 Documentation des données membres

9.7.4.1 BaseDeDonnees* PupitreConducteur : :bdd [private]

Un pointeur sur un objet de type BaseDeDonnees

Référencé par PupitreConducteur(), setIdVehicule(), verifierIdentifiantConducteur(), et visualiserInformationsArret().

9.7.4.2 QString PupitreConducteur : :codeConducteur [private]

QString contenant le code du conducteur

Référencé par gererMenuService(), prendreService(), et saisirCodeConducteur().

9.7.4.3 bool PupitreConducteur : :fini [private]

Un bool pour savoir si la gestion est fini ou non

Référencé par gererMenuService().

9.7.4.4 QString PupitreConducteur : :idVehicule [private]

QString permettant d'identifié l'id véhicule

Référencé par setIdVehicule().

9.7.4.5 QTP* PupitreConducteur : :qtp [private]

Un pointeur sur un objetde type QTP

Référencé par afficherMessage(), effacer(), gererMenuService(), PupitreConducteur(), saisirCodeConducteur(), et ~Pupitre-Conducteur().

La documentation de cette classe a été générée à partir des fichiers suivants :
– pupitreconducteur.h
– pupitreconducteur.cpp

9.8 Référence de la classe QTP

Classe de bas niveau qui gère le terminal QTP.

`#include <qtp.h>`

Fonctions membres publiques

– QTP (QString port="/dev/qtp")
  Constructeur.
– ~QTP ()
  Destructeur.
– int afficher (char *message, int nbchar)
  Affiche un message de nbchar caractères.
– char lireTouche (char *touche, int nbchar, int mode)
  Lit le code d'une touche avec ous sans le mode echo.
– void gotoxy (int ligne, int colonne)
  Déplace le curseur à la position : ligne,colonne.
– void clearEndofline ()
  Efface la fin d'une ligne.
– void clearEndofpage ()
  Efface la fin de la page.
– void cursorOn ()
  Active le curseur.
– void cursorOff ()
  Désactive le curseur.
– void cursorBlink ()
  Fait clignoter le curseur.

Fonctions membres protégées

– int send (char *cmd, int nbchar)
  Émet nbchar caractères sur le port série.
– int recv (char *mess, int nbchar)
  Reçoit nbcahr caractères du port série.
– int requestReady ()
  Émet une requête.

Attributs privés

– int _qtpid

9.8.1 Description détaillée

Auteur

Thierry Vaira <tvaira@free.fr>

9.8.2 Documentation des constructeurs et destructeur

9.8.2.1 QTP : :QTP (QString *port* = "/dev/qtp")

Paramètres

port	nom et chemin vers le fichier de périphérique

Références _qtpid.

```
{
    //Ouverture du port
    //ouvrir et configurer le port série attaché au QTP

    _qtpid = -1;
    struct termios  termios_p;

        if (( _qtpid=open(port.toLocal8Bit().constData(), O_RDWR|O_NONBLOCK ))
      == -1 )
        {
        qDebug() << Q_FUNC_INFO << "Erreur ouverture port !";
            //perror("open");
                    return;
        }

        tcgetattr(_qtpid, &termios_p);

        termios_p.c_iflag = IGNBRK | IGNPAR;
        termios_p.c_oflag = 0;
        termios_p.c_cflag = B9600 | CS8;
    termios_p.c_cflag &= ~PARENB;

        termios_p.c_lflag = ~ECHO;
        termios_p.c_cc[VMIN] = 1;
        termios_p.c_cc[VTIME] = 0;

        tcsetattr(_qtpid, TCSANOW, &termios_p);
    fcntl(_qtpid,F_SETFL,fcntl(_qtpid,F_GETFL)&~O_NONBLOCK);

    //qDebug() << Q_FUNC_INFO << "Port QTP ouvert !";
}
```

9.8.2.2 QTP : :~QTP ( )

Références _qtpid.

```
{
   //Fermeture du port
   close(_qtpid);
}
```

9.8.3 Documentation des fonctions membres

9.8.3.1 QTP : :afficher (char * *message*, int *nbchar*)

Paramètres

message	
nbchar	

Renvoie

int

Références send().

Référencé par PupitreConducteur : :afficherMessage().

```
{
```

```
        int retour;

        retour = send(message, nbchar);
        return retour;
}
```

9.8.3.2 QTP : :clearEndofline ( )

Références CLEAR_ENDOFLINE, cmds, NB_CHAR, et send().

Référencé par PupitreConducteur : :afficherMessage(), et PupitreConducteur : :effacer().

```
{
        send(cmds[CLEAR_ENDOFLINE], NB_CHAR(CLEAR_ENDOFLINE));
}
```

9.8.3.3 QTP : :clearEndofpage ( )

Références CLEAR_PAGE, cmds, NB_CHAR, et send().

```
{
        send(cmds[CLEAR_PAGE], NB_CHAR(CLEAR_PAGE));
}
```

9.8.3.4 QTP : :cursorBlink ( )

Références ABS, BLINK, cmds, NB_CHAR, et send().

```
{
        send(cmds[BLINK], NB_CHAR(ABS));
}
```

9.8.3.5 QTP : :cursorOff ( )

Références cmds, CURSOR_OFF, NB_CHAR, et send().

Référencé par PupitreConducteur : :afficherMessage(), et PupitreConducteur : :effacer().

```
{
        send(cmds[CURSOR_OFF], NB_CHAR(CURSOR_OFF));
}
```

9.8.3.6 QTP : :cursorOn ( )

Références cmds, CURSOR_ON, NB_CHAR, et send().

Référencé par PupitreConducteur : :effacer().

```
{
        send(cmds[CURSOR_ON], NB_CHAR(CURSOR_ON));
}
```

9.8.3.7 QTP : :gotoxy (int *ligne*, int *colonne*)

Paramètres

ligne	
colonne	

Références ABS, cmds, NB_CHAR, et send().

Référencé par PupitreConducteur : :afficherMessage(), et PupitreConducteur : :effacer().

```
{
        cmds[ABS][2] = (unsigned char)ligne+32;
        cmds[ABS][3] = (unsigned char)colonne+32;
        send(cmds[ABS], NB_CHAR(ABS));
}
```

9.8.3.8 QTP : :lireTouche (char * *touche*, int *nbchar*, int *mode*)

Paramètres

touche	
nbchar	
mode	

Renvoie

char

Références CR, CURSOR_DOWN, CURSOR_UP, ESC, QTP_ECHO, recv(), et send().

Référencé par PupitreConducteur : :gererMenuService(), et PupitreConducteur : :saisirCodeConducteur().

```
{
        char key;
        int lus;

        for(lus=0;lus<nbchar;lus++)
        {
                recv(&key, 1);
        if(mode == QTP_ECHO)
                {
                        if(key != CR && key != ESC && key != CURSOR_UP && key
    != CURSOR_DOWN)
                                send(&key, 1);
                }
                *(touche+lus) = key;
        }

        #ifdef DEBUG_QTP
        if(nbchar == 1)
                fprintf(stderr, "QTP::lireTouche : %c (0x%02x) -> %d\n", key,
    key, key);
        else
        {
                int i;
                fprintf(stderr, "QTP::lireTouche (%d-%d)\n", nbchar, lus);
                for(i=0;i<lus;i++)
                {
                        fprintf(stderr, "%c (0x%02x) -> %d\n", *(touche+i), *(
    touche+i), *(touche+i));
                }
        }
        #endif
        //conversion de key
    if (key == 48) // '0'
        {
        key = 0; // 0
        }
        if (key == 49)
        {
                key = 1;
        }
    if (key == 50)
        {
                key = 2;
        }
    if (key == 51)
        {
                key = 3;
        }
        if (key == 52)
        {
                key = 4;
        }
        if (key == 53)
        {
                key = 5;
        }
        if (key == 54)
        {
                key = 6;
        }
        if (key == 55)
        {
                key = 7;
        }
        if (key == 56)
        {
                key = 8;
        }
        if (key == 57)
        {
                key = 9;
```

```
        }

        return key;
}
```

9.8.3.9 QTP : :recv (char * *mess*, int *nbchar*) [protected]

Paramètres

mess	
nbchar	

Renvoie

int

Références _qtpid, CR, ERROR, FAUX, LF, MAX_LENGTH, MAX_MESS_2048, et VRAI.

Référencé par lireTouche(), et requestReady().

```
{
        int retour;
        char car;
        int lus = 0;
        int fin = FAUX;
        int MessagePresent = VRAI;

        if(_qtpid > 0 && mess != (char *)NULL)
        {
                if(nbchar > 0)
                {
                        for(lus=0;lus<nbchar;lus++)
                        {
                                retour = read(_qtpid, &car, 1);
                                if(retour > 0)
                                        *(mess+lus) = car;
                                else    break;
                        }
                        if(nbchar == lus && nbchar > 1)
                                *(mess+lus) = 0x00;//fin de chaine
                        retour = lus;
                        #ifdef DEBUG_QTP
                        int i;
                        fprintf(stderr, "QTP::_recv (%d-%d) : ", nbchar, lus);
                        for(i=0;i<lus;i++)
                                fprintf(stderr, "0x%02x ", *(mess+i));
                        fprintf(stderr, "\n");
                        #endif
                }
                else
                {
                        for(lus=0;lus<(MAX_MESS_2048*MAX_LENGTH);lus++)
                        {
                                retour = read(_qtpid, &car, 1);
                                if(car == CR)
                                        fin = VRAI;
                                if(car == LF && fin == VRAI)
                                        break;
                                if(car == (char)0xFF)
                                        MessagePresent = FAUX;
                                if(fin != VRAI)
                                        *(mess+lus) = car;
                        }
                        if(car == LF)   lus--;
                        *(mess+lus) = 0x00;//fin de chaine
                        retour = lus;
                        #ifdef DEBUG_QTP
                        int i;
                        if(MessagePresent == VRAI)
                        {
                                fprintf(stderr, "QTP::_recv (%d-%d) : ", nbchar
      , lus);
                                for(i=0;i<lus;i++)
                                        fprintf(stderr, "0x%02x ", *(mess+i));
                                fprintf(stderr, "\n");
                        }
                        else    fprintf(stderr, "QTP::_recv (%d-%d) : pas de
       message reçu !", nbchar, lus);
                        #endif
                        if(MessagePresent == FAUX)
                                retour = ERROR;
                }
        }
        else retour = _qtpid;
```

```
        return retour;
}
```

9.8.3.10 QTP : :requestReady ( ) [protected]

Renvoie

int

Références cmds, NB_CHAR, recv(), REQUEST_READY, et send().

```
{
        char etat;

        send(cmds[REQUEST_READY], NB_CHAR(REQUEST_READY));
        recv(&etat, 1);

        return (int)etat;
}
```

9.8.3.11 QTP : :send (char * *cmd,* int *nbchar*) [protected]

Paramètres

cmd	
nbchar	

Renvoie

int

Références _qtpid, et ERROR.

Référencé par afficher(), clearEndofline(), clearEndofpage(), cursorBlink(), cursorOff(), cursorOn(), gotoxy(), lireTouche(), et requestReady().

```
{
        int retour = ERROR;

        if(_qtpid > 0)
        {
                retour = write(_qtpid, cmd, nbchar);

                usleep(0);

                #ifdef DEBUG_QTP
                int i;
                //debug : affichage
                fprintf(stderr, "-> QTP::_send (%d) : ", nbchar);
                for(i=0;i<nbchar;i++)
                {
                        fprintf(stderr, "0x%02x ", *(cmd+i));
                }
                fprintf(stderr, "\n");
                #endif
        }
        else
        {
                #ifdef DEBUG_QTP
                int i;
                //debug : affichage
                fprintf(stderr, "QTP::_send (%d) : ERREUR port !", nbchar);
                for(i=0;i<nbchar;i++)
                {
                        fprintf(stderr, "0x%02x ", *(cmd+i));
                }
                fprintf(stderr, "\n");
                #endif
                retour = _qtpid;
        }

        return retour;
}
```

9.8.4 Documentation des données membres

9.8.4.1 int QTP : :_qtpid [private]

descripteur de périphérique qtp

Référencé par QTP(), recv(), send(), et ~QTP().

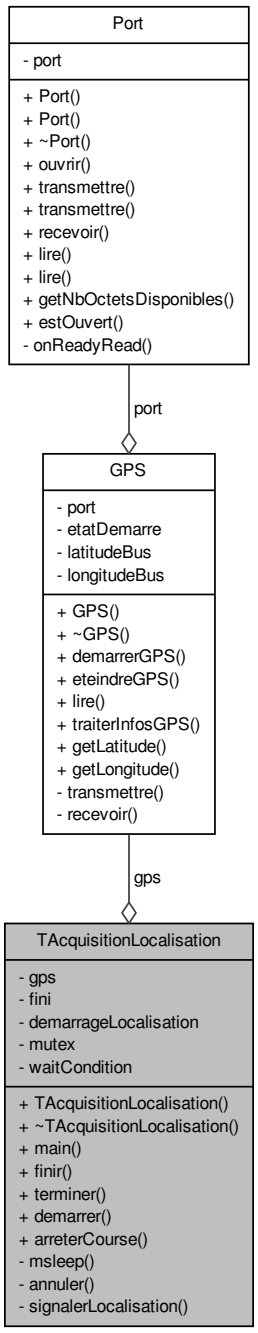
La documentation de cette classe a été générée à partir des fichiers suivants :
– qtp.h
– qtp.cpp

9.9 Référence de la classe TAcquisitionLocalisation

Classe qui permet d'acquérir la localisation et de la diffuser.

`#include <tacquisitionlocalisation.h>`

Graphe de collaboration de TAcquisitionLocalisation :



Connecteurs publics

- void main ()
  Méthode qui comprend le corps du thread de la classe TAcquisitionLocalisation.
- void finir ()
  Méthode qui permet de passer le booléen fini à l'état true.
- void terminer ()
  Méthode qui permet de terminer le thread.
- void demarrer (QString idItineraire)
  Méthode qui permet de démarrer la localisation pour une course.
- void arreterCourse (QString idItineraire)
  Méthode qui permet d'arrêter la localisation pour une course.

Signaux

- void actualiserLocalisation (QString latitudeBus, QString longitudeBus)
  Émet la localisation courante du bus.
- void actualiserLocalisation (double latitudeBus, double longitudeBus)
  Émet la localisation courante du bus.

Fonctions membres publiques

- TAcquisitionLocalisation ()
  Constructeur par défaut de la classe TAcquisitionLocalisation.
- ~TAcquisitionLocalisation ()
  Destrcuteur par défaut de la classe TAcquisitionLocalisation.

Fonctions membres privées

- void msleep (unsigned long sleepMS)
- void annuler ()
  Méthode qui permet d'annuler une temporisation en cours.
- void signalerLocalisation ()
  Méthode qui permet de récupérer et de diffuser la longitude et la latitude du bus.

Attributs privés

- GPS * gps
- bool fini
- bool demarrageLocalisation
- QMutex mutex
- QWaitCondition waitCondition

9.9.1 Description détaillée

Auteur

Rémy Roux <remy.roux84130@gmail.com>

9.9.2 Documentation des constructeurs et destructeur

9.9.2.1 TAcquisitionLocalisation : :TAcquisitionLocalisation ( )

Références demarrageLocalisation, BaseDeDonnees : :getInstance(), et gps.

```
                                                : fini(false)
{
#ifdef DEBUG_GPS
    qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
#endif
   gps = new GPS();
   demarrageLocalisation = false;
#ifdef SIMULATION_ET2
   // Simulation pour l'étudiant 2 (voir itinéaire TGestionConducteur::main())
   bdd = BaseDeDonnees::getInstance();
   qDebug() << Q_FUNC_INFO << bdd;
   bdd->connecter("referentiel-123.sqlite");
   point = 0;
   QString requete = "SELECT trace.pt_lat, trace.pt_lon  FROM trace INNER JOIN
```

```
        traces ON traces.idTrace=trace.idTrace WHERE
        traces.idItineraire='4503603922673401' ORDER BY trace.pt_sequence ASC";
    bdd->recuperer(requete, trajet);
    //qDebug() << Q_FUNC_INFO << trajet;
    etatSimulation = false;
#endif
}
```

9.9.2.2 TAcquisitionLocalisation : :~TAcquisitionLocalisation ( )

Références BaseDeDonnees : :detruireInstance(), et gps.

```
{
    delete gps;
#ifdef SIMULATION_ET2
    BaseDeDonnees::detruireInstance();
#endif
#ifdef DEBUG_GPS
    qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
#endif
}
```

9.9.3 Documentation des fonctions membres

9.9.3.1 TAcquisitionLocalisation : :actualiserLocalisation (QString *latitudeBus,* QString *longitudeBus*) [signal]

Paramètres

latitudeBus	QString
longitudeBus	QString

Référencé par signalerLocalisation().

9.9.3.2 TAcquisitionLocalisation : :actualiserLocalisation (double *latitudeBus,* double *longitudeBus*) [signal]

Paramètres

latitudeBus	double
longitudeBus	double

9.9.3.3 TAcquisitionLocalisation : :annuler ( ) [inline, private]

Références waitCondition.

Référencé par finir().

```
    {
        waitCondition.wakeAll();
    }
```

9.9.3.4 TAcquisitionLocalisation : :arreterCourse (QString *idItineraire*) [slot]

Références demarrageLocalisation, GPS : :eteindreGPS(), et gps.

```
{
    Q_UNUSED(idItineraire)

    demarrageLocalisation = false;
    gps->eteindreGPS();
}
```

9.9.3.5 TAcquisitionLocalisation : :demarrer (QString *idItineraire*) [slot]

Références demarrageLocalisation, GPS : :demarrerGPS(), et gps.

```
{
    Q_UNUSED(idItineraire)

    demarrageLocalisation = true;
    gps->demarrerGPS();
}
```

9.9.3.6 TAcquisitionLocalisation : :finir ( ) [slot]

Références annuler(), et fini.

Référencé par main().

```
{
    fini = true;
    annuler();
}
```

9.9.3.7 TAcquisitionLocalisation : :main ( ) [slot]

Références demarrageLocalisation, GPS : :eteindreGPS(), fini, gps, GPS : :lire(), msleep(), signalerLocalisation(), et GPS : :traiterInfosGPS().

```
{
#ifdef DEBUG_GPS
    qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
#endif

    //bool ok = gps->demarrerGPS();
    QStringList infosGPS;

    while(!fini)
    {
        if(demarrageLocalisation == true)
        {
            infosGPS = gps->lire();

            if(infosGPS.size() > 1)
                gps->traiterInfosGPS(infosGPS);
        }

        // Simulation pour l'étudiant 2 ?
        #ifdef SIMULATION_ET2
        simulerEtudiant2();
        #else
        if(infosGPS.size() > 1)
            signalerLocalisation();
        #endif

        this->msleep(1000); // période d'acquisition à définir
    }

    gps->eteindreGPS();
}
```

9.9.3.8 void TAcquisitionLocalisation : :msleep (unsigned long *sleepMS*) [inline, private]

Références mutex, et waitCondition.

Référencé par main().

```
    {
        waitCondition.wait(&mutex, sleepMS);
    }
```

9.9.3.9 TAcquisitionLocalisation : :signalerLocalisation ( ) [private]

Références actualiserLocalisation(), GPS : :getLatitude(), GPS : :getLongitude(), et gps.

Référencé par main().

```
{
    QString latitude;
    QString longitude;

    // récupère la localisation
    latitude = gps->getLatitude();
    longitude = gps->getLongitude();
    #ifdef DEBUG_GPS
    qDebug() << Q_FUNC_INFO << latitude << longitude;
    #endif

    // émet la localisation
    emit actualiserLocalisation(latitude, longitude);
}
```

9.9.3.10 TAcquisitionLocalisation : :terminer ( ) [slot]

```
{
#ifdef DEBUG_GPS
    qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
#endif
}
```

9.9.4 Documentation des données membres

9.9.4.1 bool TAcquisitionLocalisation : :demarrageLocalisation [private]

Un bool qui permet de démarrer la localisation ou pas

Référencé par arreterCourse(), demarrer(), main(), et TAcquisitionLocalisation().

9.9.4.2 bool TAcquisitionLocalisation : :fini [private]

Un bool qui permet de mettre fin à la boucle principale du thread

Référencé par finir(), et main().

9.9.4.3 GPS∗ TAcquisitionLocalisation : :gps [private]

Pointeur sur un objet GPS

Référencé par arreterCourse(), demarrer(), main(), signalerLocalisation(), TAcquisitionLocalisation(), et ∼TAcquisition-Localisation().

9.9.4.4 QMutex TAcquisitionLocalisation : :mutex [private]

Un mutex pour la gestion des temporisations

Référencé par msleep().

9.9.4.5 QWaitCondition TAcquisitionLocalisation : :waitCondition [private]

Pour la gestion des temporisations

Référencé par annuler(), et msleep().

La documentation de cette classe a été générée à partir des fichiers suivants :
– tacquisitionlocalisation.h
– tacquisitionlocalisation.cpp

9.10 Référence de la classe TCommunicationSAI

Classe qui permet de communiquer avec le SAI.

`#include <tcommunicationsai.h>`

Connecteurs publics

– void main ()
  Méthode qui comprend le corps du thread.
– void finir ()
  Méthode qui permet de passer le booléen fini à l'état true.
– void terminer ()
  Méthode qui permet de terminer le thread.

Fonctions membres publiques

– TCommunicationSAI ()
  Constructeur.
– ∼TCommunicationSAI ()
  Destructeur.

Fonctions membres privées

– void msleep (unsigned long sleepMS)
– void annuler ()
Méthode qui permet d'annuler une temporisation en cours.

Attributs privés

– bool fini
– QMutex mutex
– QWaitCondition waitCondition

9.10.1 Description détaillée

Auteur

Rémy Roux <remy.roux84130@gmail.com>

9.10.2 Documentation des constructeurs et destructeur

9.10.2.1 TCommunicationSAI : :TCommunicationSAI ( )

```
                                    : fini(false)
{
#ifdef DEBUG_GPS_2
    qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
#endif
}
```

9.10.2.2 TCommunicationSAI : :~TCommunicationSAI ( )

```
{
#ifdef DEBUG_GPS_2
    qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
#endif
}
```

9.10.3 Documentation des fonctions membres

9.10.3.1 TCommunicationSAI : :annuler ( ) [inline, private]

Références waitCondition.

Référencé par finir().

```
    {
        waitCondition.wakeAll();
    }
```

9.10.3.2 TCommunicationSAI : :finir ( ) [slot]

Références annuler(), et fini.

Référencé par main().

```
{
    fini = true;
    annuler();
}
```

9.10.3.3 TCommunicationSAI : :main ( ) [slot]

Références fini, et msleep().

```
{
#ifdef DEBUG_GPS_2
    qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
#endif

    while(!fini)
    {
        this->msleep(1000);
    }
}
```

9.10.3.4 void TCommunicationSAI : :msleep (unsigned long *sleepMS*) [inline, private]

Références mutex, et waitCondition.

Référencé par main().

```
    {
        waitCondition.wait(&mutex, sleepMS);
    }
```

9.10.3.5 TCommunicationSAI : :terminer ( ) [slot]

```
{
#ifdef DEBUG_GPS_2
    qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
#endif
}
```

9.10.4 Documentation des données membres

9.10.4.1 bool TCommunicationSAI : :fini [private]

Un bool qui permet de mettre fin à la boucle principale du thread

Référencé par finir(), et main().

9.10.4.2 QMutex TCommunicationSAI : :mutex [private]

Un mutex pour la gestion des temporisations

Référencé par msleep().

9.10.4.3 QWaitCondition TCommunicationSAI : :waitCondition [private]

Pour la gestion des temporisations

Référencé par annuler(), et msleep().

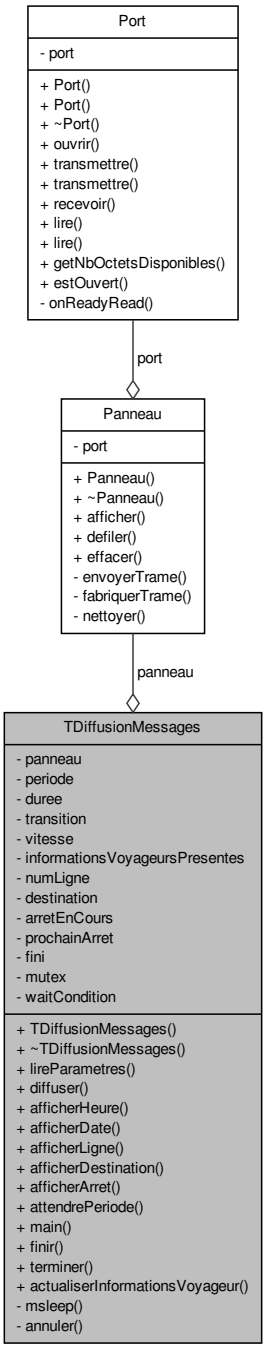
La documentation de cette classe a été générée à partir des fichiers suivants :
– tcommunicationsai.h
– tcommunicationsai.cpp

9.11 Référence de la classe TDiffusionMessages

Classe qui gère la diffussion des messages informatifs dans le bus.

`#include <tdiffusionmessages.h>`

Graphe de collaboration de TDiffusionMessages :



Connecteurs publics

- void main ()
  Le corps du thread.
- void finir ()
  Met fin à la boucle du thread.
- void terminer ()
  Termine le thread.
- void actualiserInformationsVoyageur (QString numLigne, QString destination, QString arretEnCours, QString prochainArret, int numArret)
  Actualisation des Informations Voyageurs utiles aux afficheurs.

Fonctions membres publiques

- TDiffusionMessages ()
  Constructeur de la classe TDiffusionMessages.
- ~TDiffusionMessages ()
  Destructeur de la classe TDiffusionMessages.
- void lireParametres ()
  Lecture du fichier .ini et de ses paramètres.
- void diffuser ()
  Méthode principale pour la diffusion des messages.
- void afficherHeure (int position=POS_DEFAUT+3)
  Méthode qui permet l'affichage de l'heure actuelle.
- void afficherDate (int position=POS_DEFAUT)
  Méthode qui permet l'affichage de la date courante.
- void afficherLigne (QString numLigne)
  Méthode qui permet l'affichage du numéro de Ligne.
- void afficherDestination (QString destination)
  Méthode qui permet l'affichage de la destination du bus.
- void afficherArret (QString nomArretEnCours, QString nomProchainArret)
  Méthode qui permet l'affichage de l'arrêt et du prochain arrêt du bus.
- void attendrePeriode ()
  Méthode d'attente d'une prochaine diffusion.

Fonctions membres privées

- void msleep (unsigned long sleepMS)
  Permet de faire des temporisations dans le thread.
- void annuler ()
  Annule une temporisation en cours.

Attributs privés

- Panneau panneau
- int periode
- int duree
- int transition
- int vitesse
- bool informationsVoyageursPresentes
- QString numLigne
- QString destination
- QString arretEnCours
- QString prochainArret
- bool fini
- QMutex mutex
- QWaitCondition waitCondition

9.11.1 Description détaillée

Auteur

Gabriel Destrée <gabriel.destree@gmail.com>

9.11.2 Documentation des constructeurs et destructeur

9.11.2.1 TDiffusionMessages : :TDiffusionMessages ( )

```
                                    : fini(false)
{
    #ifdef DEBUG_PANNEAU
        qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
    #endif
}
```

9.11.2.2 TDiffusionMessages : :~TDiffusionMessages ( )

```
{
    #ifdef DEBUG_PANNEAU
        qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
    #endif
}
```

9.11.3 Documentation des fonctions membres

9.11.3.1 TDiffusionMessages : :actualiserInformationsVoyageur (QString *numLigne,* QString *destination,* QString *arretEnCours,* QString *prochainArret,* int *numArret*) [slot]

Paramètres

numLigne	
destination	
arretEnCours	
prochainArret	

Références arretEnCours, destination, informationsVoyageursPresentes, numLigne, et prochainArret.

```
{
    Q_UNUSED(numArret)

    #ifdef DEBUG_PANNEAU
        qDebug() << Q_FUNC_INFO;
    #endif

    this->numLigne = numLigne;
    this->destination = destination;
    this->arretEnCours = arretEnCours;
    this->prochainArret = prochainArret;

    informationsVoyageursPresentes = true;
}
```

9.11.3.2 TDiffusionMessages : :afficherArret (QString *nomArretEnCours,* QString *nomProchainArret*)

Paramètres

nomArretEnCours	
nomProchainArret	

Références Panneau : :afficher(), Panneau : :defiler(), duree, MAX_TEXTE, panneau, et vitesse.

Référencé par diffuser().

```
{
    QString messageArretEnCours = "Arret : " + nomArretEnCours;
    QString messageProchainArret = "Prochain Arret : " + nomProchainArret;

    if(messageArretEnCours.length() > MAX_TEXTE)
        panneau.defiler(messageArretEnCours, vitesse);
    else
        panneau.afficher(messageArretEnCours, duree);

    if(!nomProchainArret.isEmpty()) // le bus arrive arrive au terminus
    {
        if(messageProchainArret.length() > MAX_TEXTE)
            panneau.defiler(messageProchainArret, vitesse);
        else
            panneau.afficher(messageProchainArret, duree);
    }
}
```

9.11.3.3 TDiffusionMessages : :afficherDate (int *position* = POS_DEFAUT)

Paramètres

position	du premier caractère du message sur le Panneau

Références Panneau : :afficher(), duree, et panneau.

Référencé par diffuser().

```
{
    QDateTime dateActuelle = QDateTime::currentDateTime(); // appel méthode
       statique qui retourne l'heure et la date actuelle
    QString date = dateActuelle.toString("dd/MM/yyyy");

    panneau.afficher(date, duree, position);
}
```

9.11.3.4 TDiffusionMessages : :afficherDestination (QString *destination*)

Paramètres

destination	

Références Panneau : :afficher(), Panneau : :defiler(), destination, duree, MAX_TEXTE, panneau, et vitesse.

```
{
    QString messageDestination = "Destination : " + destination;
    //destination = "Gare Avignon TGV";

    // fixe ou défilant ?
    if(messageDestination.length() > MAX_TEXTE)
        panneau.defiler(messageDestination, vitesse);
    else
        panneau.afficher(messageDestination, duree);
}
```

9.11.3.5 TDiffusionMessages : :afficherHeure (int *position* = POS_DEFAUT+3)

Paramètres

position	du premier caractère du message sur le Panneau

Références Panneau : :afficher(), duree, et panneau.

Référencé par diffuser().

```
{
    QDateTime heureActuelle = QDateTime::currentDateTime(); // appel méthode
       statique qui retourne l'heure et la date actuelle
    QString heure = heureActuelle.toString("hh:mm");

    panneau.afficher(heure, duree, position);
}
```

9.11.3.6 TDiffusionMessages : :afficherLigne (QString *numLigne*)

Paramètres

numLigne	

Références Panneau : :afficher(), Panneau : :defiler(), duree, MAX_TEXTE, numLigne, panneau, et vitesse.

Référencé par diffuser().

```
{
    QString messageLigne = "Ligne : " + numLigne;

    // fixe ou défilant ?
    if(messageLigne.length() > MAX_TEXTE)
        panneau.defiler(messageLigne, vitesse);
    else
        panneau.afficher(messageLigne, duree);
}
```

9.11.3.7 TDiffusionMessages : :annuler ( ) [inline, private]

Références waitCondition.

Référencé par finir().

```
    {
        waitCondition.wakeAll();
    }
```

9.11.3.8 TDiffusionMessages : :attendrePeriode ( )

Références msleep(), et periode.

Référencé par main().

```
{
    //attente prochaine diffusion
    this->msleep(periode);
}
```

9.11.3.9 TDiffusionMessages : :diffuser ( )

Références afficherArret(), afficherDate(), afficherHeure(), afficherLigne(), arretEnCours, Panneau : :effacer(), informationsVoyageursPresentes, numLigne, panneau, prochainArret, et transition.

Référencé par main().

```
{
        afficherHeure();
        panneau.effacer(transition);

        afficherDate();
        panneau.effacer(transition);

    if(informationsVoyageursPresentes == true)
    {
         afficherLigne(numLigne);
         panneau.effacer(transition);

         //afficherDestination(destination);
         panneau.effacer(transition);

         afficherArret(arretEnCours, prochainArret);
         panneau.effacer(transition);
    }
}
```

9.11.3.10 TDiffusionMessages : :finir ( ) [slot]

Références annuler(), et fini.

Référencé par TInformationVoyageur : :main().

```
{
    fini = true;
    annuler();
}
```

9.11.3.11 TDiffusionMessages : :lireParametres ( )

Références duree, periode, transition, et vitesse.

Référencé par main().

```
{
    // Le nom du fichier INI : nom-executable.ini
    QString fichierINI = qApp->applicationName() + ".ini"; // siv.ini

    QSettings parametres(fichierINI, QSettings::IniFormat);

    // Lecture des paramètres de configuration
    periode = parametres.value("diffusion/periode", "1000").toInt(); // en ms

    // durée
    duree = parametres.value("diffusion/duree", "2").toInt();

    // transition
    transition = parametres.value("diffusion/transition", "1").toInt();

    // vitesse
    vitesse = parametres.value("diffusion/vitesse", "250").toInt();

    // Affichage de débugage
    #ifdef DEBUG_PANNEAU
    qDebug() << Q_FUNC_INFO << QString::fromUtf8("période : %1").arg(periode);
    qDebug() << Q_FUNC_INFO << QString::fromUtf8("duree : %1").arg(duree);
    qDebug() << Q_FUNC_INFO << QString::fromUtf8("transition : %1").arg(
      transition);
    qDebug() << Q_FUNC_INFO << QString::fromUtf8("vitesse : %1").arg(vitesse);
    #endif
}
```

9.11.3.12 TDiffusionMessages : :main ( ) [slot]

Références attendrePeriode(), diffuser(), fini, informationsVoyageursPresentes, et lireParametres().

```
{

#ifdef DEBUG_PANNEAU
    qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << "demarrage" <<
```

```
      this;
#endif

    lireParametres();

    // Diffusion périodique des messages

    informationsVoyageursPresentes = false;
    while(!fini)
    {
        diffuser();

        attendrePeriode();
    }

    #ifdef DEBUG_PANNEAU
        qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << "arret" <<
      this;
    #endif
}
```

9.11.3.13 TDiffusionMessages : :msleep (unsigned long *sleepMS*) [inline, private]

Paramètres

sleepMS	

Références mutex, et waitCondition.

Référencé par attendrePeriode().

```
    {
        waitCondition.wait(&mutex, sleepMS);
    }
```

9.11.3.14 TDiffusionMessages : :terminer ( ) [slot]

```
{
    #ifdef DEBUG_PANNEAU
        qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
    #endif
}
```

9.11.4 Documentation des données membres

9.11.4.1 QString TDiffusionMessages : :arretEnCours [private]

Un QString contenant le nom de l'arrêt en cours

Référencé par actualiserInformationsVoyageur(), et diffuser().

9.11.4.2 QString TDiffusionMessages : :destination [private]

Un QString contenant le nom de la destination

Référencé par actualiserInformationsVoyageur(), et afficherDestination().

9.11.4.3 int TDiffusionMessages : :duree [private]

Un int correspondant à la durée d'affichage d'un message

Référencé par afficherArret(), afficherDate(), afficherDestination(), afficherHeure(), afficherLigne(), et lireParametres().

9.11.4.4 bool TDiffusionMessages : :fini [private]

Un bool permettant de savoir si le thread est terminé

Référencé par finir(), et main().

9.11.4.5 bool TDiffusionMessages : :informationsVoyageursPresentes [private]

Un bool permettant de savoir si les informations voyageurs sont présentes ou non

Référencé par actualiserInformationsVoyageur(), diffuser(), et main().

9.11.4.6 QMutex TDiffusionMessages : :mutex [private]

mutex pour la gestion des temporisations

Référencé par msleep().

9.11.4.7 QString TDiffusionMessages : :numLigne [private]

Un QString contenant le numéro de ligne

Référencé par actualiserInformationsVoyageur(), afficherLigne(), et diffuser().

9.11.4.8 Panneau TDiffusionMessages : :panneau [private]

Objet panneau de type Panneau

Référencé par afficherArret(), afficherDate(), afficherDestination(), afficherHeure(), afficherLigne(), et diffuser().

9.11.4.9 int TDiffusionMessages : :periode [private]

Un int correspondant à la période d'une diffusion complète

Référencé par attendrePeriode(), et lireParametres().

9.11.4.10 QString TDiffusionMessages : :prochainArret [private]

Un QString contenant le nom du prochain arrêt

Référencé par actualiserInformationsVoyageur(), et diffuser().

9.11.4.11 int TDiffusionMessages : :transition [private]

Un int correspondant à la durée de maintien de l'effacement

Référencé par diffuser(), et lireParametres().

9.11.4.12 int TDiffusionMessages : :vitesse [private]

Un int correspondant à la vitesse de défilement

Référencé par afficherArret(), afficherDestination(), afficherLigne(), et lireParametres().

9.11.4.13 QWaitCondition TDiffusionMessages : :waitCondition [private]

pour la gestion des temporisations

Référencé par annuler(), et msleep().

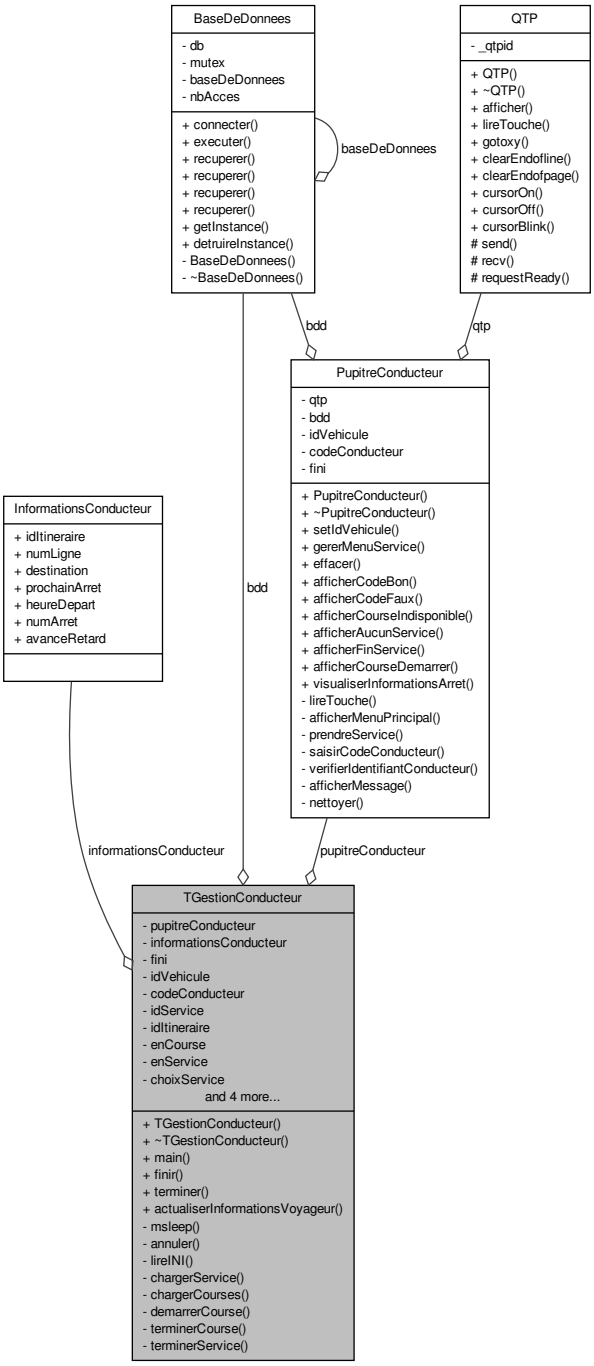
La documentation de cette classe a été générée à partir des fichiers suivants :
– tdiffusionmessages.h
– tdiffusionmessages.cpp

9.12 Référence de la classe TGestionConducteur

Classe qui gère la gestion du conducteur : Prise de service, prendre une courses, afficher l'itinéraire..

`#include <tgestionconducteur.h>`

Graphe de collaboration de TGestionConducteur :



Connecteurs publics

- void main ()
  Le corp du thread.
- void finir ()
  Met fin à la boucle du thread.
- void terminer ()
  Met fin au thread.
- void actualiserInformationsVoyageur (QString numLigne, QString destination, QString arretEnCours, QString prochainArret, int numeroArret)
  Slot déclenché par le signal envoyé par TInformationVoyageur.

Signaux

- void actualiserCourse (QString numItineraire)
  Signal actualisant les informations de la course en cours.
- void courseFinie (QString idItineraire)
  Signal indiquant qu'une course est finie pour le conducteur.
- void debutService (QString codeConducteur)
  Signal indiquant la prise de service du conducteur.
- void finService (QString codeConducteur)
  Signal indiquant la fin de service du conducteur.

Fonctions membres publiques

- TGestionConducteur ()
  Constructeur par défaut de la classe TGestionConducteur.
- ~TGestionConducteur ()
  Destructeur de la classe TGestionConducteur.

Fonctions membres privées

- void msleep (unsigned long sleepMS)
- void annuler ()
  Annule une temporisation en cours.
- void lireINI ()
  Permet de lire dans un fichier .ini.
- bool chargerService ()
  Permet de charger le service d'un conducteur.
- bool chargerCourses ()
  Permet de charger les courses d'un service.
- bool demarrerCourse ()
  Permet de charger la course à effectuer.
- void terminerCourse ()
  Met fin à la course en cours.
- void terminerService ()
  Met fin au service en cours.

Attributs privés

- PupitreConducteur pupitreConducteur
- InformationsConducteur informationsConducteur
- bool fini
- QString idVehicule
- QString codeConducteur
- QString idService
- QString idItineraire
- int enCourse
- int enService
- int choixService
- QStringList courseEnCours
- QVector< QStringList > courses
- BaseDeDonnees * bdd
- QMutex mutex
- QWaitCondition waitCondition

9.12.1 Description détaillée

Auteur

Cyril Cambe <cambecyril@yahoo.fr>

9.12.2 Documentation des constructeurs et destructeur

9.12.2.1 TGestionConducteur : :TGestionConducteur ( )

Références InformationsConducteur : :avanceRetard, bdd, BaseDeDonnees : :connecter(), InformationsConducteur : :destination, BaseDeDonnees : :getInstance(), InformationsConducteur : :heureDepart, InformationsConducteur : :idItineraire, idVehicule, informationsConducteur, lireINI(), InformationsConducteur : :numArret, InformationsConducteur : :numLigne, InformationsConducteur : :prochainArret, pupitreConducteur, et PupitreConducteur : :setIdVehicule().

```
                                        : fini(false), idVehicule(""),
      codeConducteur(""),idItineraire(""), enCourse(PAS_EN_COURSE), enService(
      PAS_EN_SERVICE), choixService(AUCUN), courseEnCours(QString("")), bdd(NULL)
{
    #ifdef DEBUG_QTP
    qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
    #endif
    lireINI();

    informationsConducteur.idItineraire = "";
    informationsConducteur.numLigne = "";
    informationsConducteur.destination = "";
    informationsConducteur.prochainArret = "";
    informationsConducteur.heureDepart = "";
    informationsConducteur.numArret = "";
    informationsConducteur.avanceRetard = 0;

    bdd = BaseDeDonnees::getInstance();
    if(!idVehicule.isEmpty())
        bdd->connecter("referentiel-" + idVehicule + ".sqlite");

    pupitreConducteur.setIdVehicule(idVehicule);
}
```

9.12.2.2 TGestionConducteur : :~TGestionConducteur ( )

```
{
    #ifdef DEBUG_QTP
    qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
    #endif
}
```

9.12.3 Documentation des fonctions membres

9.12.3.1 TGestionConducteur : :actualiserCourse (QString *numItineraire*) [signal]

Paramètres

numItineraire	

Référencé par demarrerCourse().

9.12.3.2 TGestionConducteur : :actualiserInformationsVoyageur (QString *numLigne,* QString *destination,* QString *arretEnCours,* QString *prochainArret,* int *numeroArret*) [slot]

Paramètres

numLigne	
destination	
arretEnCours	
prochainArret	

Références InformationsConducteur : :destination, enCourse, InformationsConducteur : :idItineraire, informationsConducteur, InformationsConducteur : :numArret, InformationsConducteur : :numLigne, PAS_EN_COURSE, InformationsConducteur-: :prochainArret, pupitreConducteur, et PupitreConducteur : :visualiserInformationsArret().

```
{
    #ifdef DEBUG_QTP
    qDebug() << Q_FUNC_INFO << numLigne << destination << arretEnCours <<
      prochainArret << numeroArret;
    #endif
    QString numArret = QString::number(numeroArret);
    informationsConducteur.numLigne = numLigne;
    informationsConducteur.destination = destination;
    informationsConducteur.prochainArret = prochainArret;
    informationsConducteur.numArret = numArret;
    pupitreConducteur.visualiserInformationsArret(informationsConducteur.
      idItineraire, informationsConducteur.numLigne, informationsConducteur.
      destination, informationsConducteur.prochainArret, informationsConducteur.
      numArret);
    if(informationsConducteur.prochainArret.length() == 0 && numArret > 0)
    {
        enCourse = PAS_EN_COURSE;
    }
}
```

9.12.3.3 TGestionConducteur : :annuler ( ) [inline, private]

Références waitCondition.

Référencé par finir().

```
{
    waitCondition.wakeAll();
}
```

9.12.3.4 TGestionConducteur : :chargerCourses ( ) [private]

Renvoie

bool

Références bdd, courses, idService, et BaseDeDonnees : :recuperer().

Référencé par main().

```
{
    QString requete = "SELECT course.idItineraire, nomConducteur, destination,
       direction, itineraire.idLigne, heureDepart, nomLong, nomCourt, description,
       course.effectue FROM service INNER JOIN conducteur ON
       conducteur.codeConducteur=service.codeConducteur INNER JOIN course ON course.idService=service.idService INNER
       JOIN itineraire ON itineraire.idItineraire=course.idItineraire INNER JOIN arret
       ON arret.idItineraire=itineraire.idItineraire INNER JOIN ligne ON
       ligne.idLigne=itineraire.idLigne WHERE service.idService='" + idService + "' AND
       course.effectue=0 AND arret.numeroSequence='0' ORDER BY heureDepart ASC";
    QVector<QStringList> lesCourses;
    bool etat = bdd->recuperer(requete, lesCourses);

    if(etat == true)
    {
        courses = lesCourses;
        #ifdef DEBUG_QTP_2
        qDebug() << Q_FUNC_INFO << courses << endl;
        #endif
        return true;
    }
    else
        return false;
}
```

9.12.3.5 TGestionConducteur : :chargerService ( ) [private]

Renvoie

bool

Références bdd, codeConducteur, BaseDeDonnees : :executer(), idService, et BaseDeDonnees : :recuperer().

Référencé par main().

```
{
    QString numeroService;
    QString requete = "SELECT idService FROM service WHERE codeConducteur='" +
      codeConducteur + "' AND effectue=0";

    bool etat = bdd->recuperer(requete, numeroService);

    if(etat == true)
    {
        idService = numeroService;

        QDateTime maintenant = QDateTime::currentDateTime();
        requete = "UPDATE service SET dateDebut='" + maintenant.toString("
      yyyy-MM-dd") + "',heureDebut='" + maintenant.toString("hh:mm:ss") + "' WHERE
       idService =" + idService;
        bool retour = bdd->executer(requete);
        if(retour == false)
        {
            #ifdef DEBUG_QTP
            qDebug() << Q_FUNC_INFO << QString::fromUtf8("erreur modification
      table course !");
            #endif
        }
        return true;
    }
    else
        return false;
}
```

9.12.3.6 TGestionConducteur : :courseFinie (QString *idItineraire*) [signal]

Paramètres

idItineraire	

Référencé par terminerCourse().

9.12.3.7 TGestionConducteur : :debutService (QString *codeConducteur*) [signal]

Paramètres

codeConducteur	

Référencé par main().

9.12.3.8 TGestionConducteur : :demarrerCourse ( ) [private]

Renvoie

bool

Références actualiserCourse(), courseEnCours, courses, InformationsConducteur : :idItineraire, idItineraire, informationsConducteur, et terminerCourse().

Référencé par main().

```
{
    QStringList uneCourse;
    bool trouve = false;

    // avant de demarrer une course il faut mettre fin à la precedente
    terminerCourse();

    for(int i=0; (i < courses.size()) && (trouve == false); i++)
    {
        uneCourse = courses.at(i);
        #ifdef DEBUG_QTP
        qDebug() << Q_FUNC_INFO << "uneCourse :" << uneCourse << endl;
        #endif
        // non effectue ?
        if(uneCourse.at(9) == "0")
        {
            trouve = true;
        }
    }

    if(trouve == true)
    {
        courseEnCours = uneCourse;
        idItineraire = courseEnCours.at(0);
        informationsConducteur.idItineraire = idItineraire;
        //pupitreConducteur.setIdItineraire(idItineraire);
        #ifdef DEBUG_QTP
        qDebug() << Q_FUNC_INFO << "idItineraire = " << idItineraire;
        #endif

        emit actualiserCourse(idItineraire);
    }
    else
    {
        courseEnCours.clear();
        courseEnCours << "";
        #ifdef DEBUG_QTP
        qDebug() << Q_FUNC_INFO  << QString::fromUtf8("plus de course à
       effectuer !");
        #endif
    }

    #ifdef DEBUG_QTP
    qDebug() << Q_FUNC_INFO << "courseEncours :" << courseEnCours << endl;
    #endif

    return trouve;
}
```

9.12.3.9 TGestionConducteur : :finir ( ) [slot]

Références annuler(), et fini.

Référencé par main().

```
{
    fini = true;
    annuler();
}
```

9.12.3.10 TGestionConducteur : :finService (QString *codeConducteur*) [signal]

Paramètres

codeConducteur	

Référencé par main().

9.12.3.11 TGestionConducteur : :lireINI ( ) [private]

Références idVehicule.

Référencé par TGestionConducteur().

```
{
    // Le nom du fichier INI : nom-executable.ini
    QString fichierINI = qApp->applicationName() + ".ini";

    QSettings parametres(fichierINI, QSettings::IniFormat);

    // Lecture des paramètres de configuration
    idVehicule = parametres.value("siv/bus","").toString();

    // Affichage de débugage
    qDebug()<<QString::fromUtf8("numéro de bus : %1").arg(idVehicule);
}
```

9.12.3.12 TGestionConducteur : :main ( ) [slot]

Références AFFICHER_COURSE, PupitreConducteur : :afficherCourseDemarrer(), PupitreConducteur : :afficherCourseIndisponible(), chargerCourses(), chargerService(), choixService, codeConducteur, debutService(), DEMARRER_COURSE, demarrerCourse(), InformationsConducteur : :destination, PupitreConducteur : :effacer(), EN_COURSE, EN_SERVICE, enCourse, enService, FIN_SERVICE, fini, finService(), PupitreConducteur : :gererMenuService(), InformationsConducteur : :idItineraire, idService, informationsConducteur, InformationsConducteur : :numArret, InformationsConducteur : :numLigne, PAS_EN_COURSE, PAS_EN_SERVICE, PRISE_SERVICE, InformationsConducteur : :prochainArret, pupitreConducteur, terminerService(), et PupitreConducteur : :visualiserInformationsArret().

```
{
    #ifdef DEBUG_QTP
    qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
    #endif

    // Communication avec l'étudiant 2
    //emit actualiserCourse("4503603922673401");

    while(!fini)
    {

        #ifdef DEBUG_QTP
        qDebug() << Q_FUNC_INFO << "enService" << enService;
        #endif
        choixService = pupitreConducteur.gererMenuService(codeConducteur,
      enService, enCourse);
        #ifdef DEBUG_QTP
        qDebug() << Q_FUNC_INFO << "choixService" << choixService;
        #endif

        if(choixService == PRISE_SERVICE && enService == PAS_EN_SERVICE)
        {
            #ifdef DEBUG_QTP
            qDebug() << Q_FUNC_INFO << QThread::currentThreadId() <<"
      choixServce :" << choixService << "codeConducteur :" <<codeConducteur;
            #endif
            enService = EN_SERVICE;
            chargerService();
            emit debutService(codeConducteur);
            #ifdef DEBUG_QTP
            qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << "idService
      " << idService << "codeConducteur" << codeConducteur;
            #endif
        }

        if(choixService == DEMARRER_COURSE && enService == EN_SERVICE)
        {
```

```
            #ifdef DEBUG_QTP
            qDebug() << Q_FUNC_INFO << QThread::currentThreadId() <<"
      choixServce :" << choixService << "codeConducteur :" <<codeConducteur;
            #endif
            enCourse = EN_COURSE;
            chargerCourses();
            if(!demarrerCourse())
            {
                pupitreConducteur.afficherCourseIndisponible();
            }
            else
            {
                pupitreConducteur.afficherCourseDemarrer();
                choixService = AFFICHER_COURSE;
            }
        }

        if(choixService == FIN_SERVICE && enService == EN_SERVICE)
        {
            enCourse = PAS_EN_COURSE;
            enService = PAS_EN_SERVICE;
            terminerService();
            emit finService(codeConducteur);
        }
        if(choixService == AFFICHER_COURSE && enCourse == EN_COURSE)
        {
            pupitreConducteur.visualiserInformationsArret(informationsConducteur
      .idItineraire, informationsConducteur.numLigne, informationsConducteur.
      destination, informationsConducteur.prochainArret, informationsConducteur.
      numArret);
        }
    }

    pupitreConducteur.effacer();
}
```

9.12.3.13 void TGestionConducteur : :msleep (unsigned long *sleepMS*) [inline, private]

Références mutex, et waitCondition.

```
    {
        waitCondition.wait(&mutex, sleepMS);
    }
```

9.12.3.14 TGestionConducteur : :terminer ( ) [slot]

```
{
    #ifdef DEBUG_QTP
    qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
    #endif
}
```

9.12.3.15 TGestionConducteur : :terminerCourse ( ) [private]

Renvoie

Références bdd, courseEnCours, courseFinie(), courses, BaseDeDonnees : :executer(), et idItineraire.

Référencé par demarrerCourse(), et terminerService().

```
{
    QString idItineraire;
    QStringList uneCourse;
    bool trouve = false;

    idItineraire = courseEnCours.at(0);

    // recherche la course à terminer
    for(int i=0; i < courses.size() && trouve == false; i++)
    {
        uneCourse = courses.at(i);
        // non effectue ?
        if(uneCourse.at(0) == idItineraire)
        {
            courses[i][9] = "1"; // course effectuee // 9 : element bdd
       "effectuee"
            trouve = true;
        }
    }
```

```
    if(trouve == true)
    {
        // on met à jour la base de données
        QString requete = "UPDATE course SET effectue=1 WHERE idItineraire='" +
       idItineraire + "'" ;
        bool retour = bdd->executer(requete);
        if(retour == false)
        {
            qDebug() << Q_FUNC_INFO << QString::fromUtf8("erreur modification
       table course !");
        }
        else
        {
            #ifdef DEBUG_QTP
            qDebug() << Q_FUNC_INFO << "course effectuee :" << idItineraire;
            #endif
            emit courseFinie(idItineraire);
        }
    }
}
```

9.12.3.16 TGestionConducteur : :terminerService ( ) [private]

Renvoie

Références PupitreConducteur : :afficherFinService(), bdd, BaseDeDonnees : :executer(), idService, pupitreConducteur, et terminerCourse().

Référencé par main().

```
{
    QDateTime maintenant = QDateTime::currentDateTime();

    //On termine la dernière course avant la fin de service
    terminerCourse();

    //on met à jour la base de données de fin de service
    QString requete = "UPDATE service SET effectue=1,dateFin='" + maintenant.
      toString("yyyy-MM-dd") + "',heureFin='" + maintenant.toString("hh:mm:ss") + "'
       WHERE idService ='" + idService +"'";
    bool retour = bdd->executer(requete);
    if(retour == false)
    {
        qDebug() << Q_FUNC_INFO << QString::fromUtf8("erreur modification table
       service !");
    }

    pupitreConducteur.afficherFinService();
}
```

9.12.4 Documentation des données membres

9.12.4.1 BaseDeDonnees* TGestionConducteur : :bdd [private]

Pointeur bdd sur la base de donnée

Référencé par chargerCourses(), chargerService(), terminerCourse(), terminerService(), et TGestionConducteur().

9.12.4.2 int TGestionConducteur : :choixService [private]

int permettant de savoir quel est le service choisi par le conducteur

Référencé par main().

9.12.4.3 QString TGestionConducteur : :codeConducteur [private]

QString contenant le code du conducteur

Référencé par chargerService(), et main().

9.12.4.4 QStringList TGestionConducteur : :courseEnCours [private]

Référencé par demarrerCourse(), et terminerCourse().

9.12.4.5 QVector<QStringList> TGestionConducteur : :courses [private]

StringList contenant le course en cours Vector de Stringlist contenant la liste des courses d'un service

Référencé par chargerCourses(), demarrerCourse(), et terminerCourse().

9.12.4.6 int TGestionConducteur : :enCourse [private]

int permettant de savoir si le conducteur est en course, pas en course ou si la course est terminée

Référencé par actualiserInformationsVoyageur(), et main().

9.12.4.7 int TGestionConducteur : :enService [private]

int permettant de savoir si le conducteur est en service ou non

Référencé par main().

9.12.4.8 bool TGestionConducteur : :fini [private]

Un bool pour savoir si la gestion est fini ou non

Référencé par finir(), et main().

9.12.4.9 QString TGestionConducteur : :idItineraire [private]

QString contenant le numéro d'itinéraire en cours

Référencé par demarrerCourse(), et terminerCourse().

9.12.4.10 QString TGestionConducteur : :idService [private]

QString contenant le numéro de service en cours

Référencé par chargerCourses(), chargerService(), main(), et terminerService().

9.12.4.11 QString TGestionConducteur : :idVehicule [private]

QString permettant d'identifié l'id véhicule

Référencé par lireINI(), et TGestionConducteur().

9.12.4.12 InformationsConducteur TGestionConducteur : :informationsConducteur [private]

pour l'affichage pupitre

Référencé par actualiserInformationsVoyageur(), demarrerCourse(), main(), et TGestionConducteur().

9.12.4.13 QMutex TGestionConducteur : :mutex [private]

Mutex pour la gestion des temporisation

Référencé par msleep().

9.12.4.14 PupitreConducteur TGestionConducteur : :pupitreConducteur [private]

Objet pupitreConducteur de type PupitreConducteur

Référencé par actualiserInformationsVoyageur(), main(), terminerService(), et TGestionConducteur().

9.12.4.15 QWaitCondition TGestionConducteur : :waitCondition [private]

Pour la gestion des temporisation

Référencé par annuler(), et msleep().

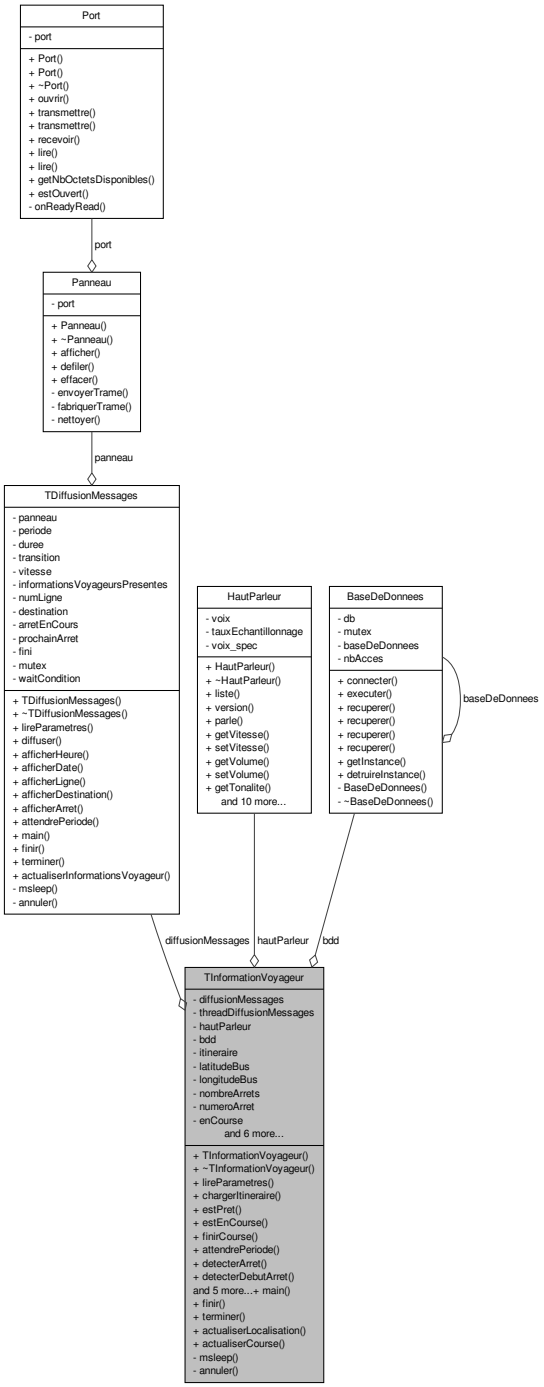
La documentation de cette classe a été générée à partir des fichiers suivants :
- tgestionconducteur.h
- tgestionconducteur.cpp

9.13 Référence de la classe TInformationVoyageur

Classe qui informe les voyageurs et qui gère la detection des arrêts.

`#include <tinformationvoyageur.h>`

Graphe de collaboration de TInformationVoyageur :



Connecteurs publics

– void main ()
  Le corps du thread.
– void finir ()
  Met fin à la boucle du thread.
– void terminer ()

Termine le thread.
- void actualiserLocalisation (QString latitude, QString longitude)
  Actualisation de la position GPS du bus.
- void actualiserCourse (QString numItineraire)
  Actualisation d'une course utile à la detection des arrêts.

Signaux

- void actualiserInformationsVoyageur (QString numLigne, QString destination, QString arretEnCours, QString prochainArret, int numeroArret)
  Signal qui envoie les informations déduites des méthodes de detection des arrêts.

Fonctions membres publiques

- TInformationVoyageur ()
  Constructeur de la classe TInformationVoyageur.
- ~TInformationVoyageur ()
  Destruteur de la classe TInformationVoyageur.
- void lireParametres ()
  Lecture du fichier .ini et de ses paramètres.
- void chargerItineraire (QString numItineraire)
  Charge l'itineraire avec une requête SQL.
- bool estPret ()
  Permet de savoir si le bus est prêt.
- int estEnCourse ()
  Permet de savoir si le bus est sur le parcours.
- bool finirCourse ()
  Permet de savoir si le bus a terminé sa course ou non.
- void attendrePeriode ()
  Attente d'une periode avant réactualisation.
- void detecterArret ()
  Methode principale de detection des arrêts.
- bool detecterDebutArret ()
  Detecte lorsque le bus entre dans la zone d'arrêt.
- double calculerDistance (QString latitudeBus, QString longitudeBus, QString latitudeArret, QString longitudeArret)
  Calcul la distance entre le bus et le rayon de detection.
- void declencherDebutArret ()
  Le bus est entré dans la zone d'arrêt.
- void setZoneArret (bool etat)
  Permet de savoir concretement si le bus est dans la zone d'arrêt.
- void signalerArret ()
  Appelle des fonctions du haut parleur pour l'énonciation des arrêts.
- bool detecterFinArret ()
  Detecte lorsque le bus sort de la zone d'arrêt.
- void declencherFinArret ()
  Le bus est sorti de la zone d'arrêt.

Fonctions membres privées

- void msleep (unsigned long sleepMS)
  Permet de faire des temporisations dans le thread.
- void annuler ()
  Annule une temporisation en cours.

Attributs privés

- TDiffusionMessages diffusionMessages
- QThread threadDiffusionMessages
- HautParleur * hautParleur
- BaseDeDonnees * bdd
- QVector< QStringList > itineraire
- QString latitudeBus
- QString longitudeBus
- int nombreArrets
- int numeroArret
- int enCourse
- int periode
- double rayon
- bool pret

– bool arret
– bool fini
– QMutex mutex
– QWaitCondition waitCondition

9.13.1 Description détaillée

Auteur

Gabriel Destrée <gabriel.destree@gmail.com>

9.13.2 Documentation des constructeurs et destructeur

9.13.2.1 TInformationVoyageur : :TInformationVoyageur ( )

Références actualiserInformationsVoyageur(), arret, bdd, BaseDeDonnees : :connecter(), diffusionMessages, enCourse, fini, -BaseDeDonnees : :getInstance(), hautParleur, main(), nombreArrets, numeroArret, PAS_EN_COURSE, periode, pret, rayon, terminer(), et threadDiffusionMessages.

```
                                        : fini(false)
{
    hautParleur = new HautParleur;

    #ifdef DEBUG_PANNEAU
        qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
    #endif

    diffusionMessages.moveToThread(&threadDiffusionMessages);
    QObject::connect(this, SIGNAL(actualiserInformationsVoyageur(QString,
      QString,QString,QString,int)), &diffusionMessages, SLOT(actualiserInformationsVoyageur
      (QString,QString,QString,QString,int)), Qt::DirectConnection);
    QObject::connect(&threadDiffusionMessages, SIGNAL(started()), &
      diffusionMessages, SLOT(main()));
    QObject::connect(&threadDiffusionMessages, SIGNAL(finished()), &
      diffusionMessages, SLOT(terminer()));

    numeroArret = -1;
    nombreArrets = 0;
    enCourse = PAS_EN_COURSE;
    periode = 0.;
    rayon   = 0.025; // 20 m
    pret    = false;
    arret   = false;
    fini    = false;
    bdd = BaseDeDonnees::getInstance();
    bdd->connecter("referentiel-123.sqlite");
}
```

9.13.2.2 TInformationVoyageur : :~TInformationVoyageur ( )

Références BaseDeDonnees : :detruireInstance(), et hautParleur.

```
{
    delete hautParleur;
    BaseDeDonnees::detruireInstance();
    #ifdef DEBUG_PANNEAU
        qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
    #endif
}
```

9.13.3 Documentation des fonctions membres

9.13.3.1 TInformationVoyageur : :actualiserCourse (QString *numItineraire*) [slot]

Paramètres

numItineraire	

Références chargerItineraire().

```
{
    #ifdef DEBUG_PANNEAU
    qDebug() << Q_FUNC_INFO << numItineraire;
    #endif
```

```
    chargerItineraire(numItineraire);
}
```

9.13.3.2 TInformationVoyageur : :actualiserInformationsVoyageur (QString *numLigne,* QString *destination,* QString *arretEnCours,* QString *prochainArret,* int *numeroArret*) [signal]

Paramètres

numLigne	
destination	
arretEnCours	
prochainArret	

Référencé par signalerArret(), et TInformationVoyageur().

9.13.3.3 TInformationVoyageur : :actualiserLocalisation (QString *latitude,* QString *longitude*) [slot]

Paramètres

latitude	
longitude	

Références latitudeBus, et longitudeBus.

```
{
    #ifdef DEBUG_PANNEAU
    qDebug() << Q_FUNC_INFO << latitude << "," << longitude;
    #endif

    // Mise à jour de la localisation du bus

    latitudeBus = latitude;
    longitudeBus = longitude;
}
```

9.13.3.4 TInformationVoyageur : :annuler ( ) [inline, private]

Références waitCondition.

Référencé par finir().

```
    {
        waitCondition.wakeAll();
    }
```

9.13.3.5 TInformationVoyageur : :attendrePeriode ( )

Références msleep(), et periode.

Référencé par main().

```
{
    //attente prochaine diffusion
    this->msleep(periode);
}
```

9.13.3.6 TInformationVoyageur : :calculerDistance (QString *latitudeBus,* QString *longitudeBus,* QString *latitudeArret,* QString *longitudeArret*)

Paramètres

latitudeBus	
longitudeBus	
latitudeArret	
longitudeArret	

Renvoie

double

Références PI.

Référencé par detecterDebutArret(), et detecterFinArret().

```
{
    double rayonTerreRadians = 6378137;   // Terre = sphère de 6378km de rayon
    double distanceLatitude = 0.;
    double distanceLongitude = 0.;
    double distance = 0.;

    double latitudeBusDouble = 0.;
    double longitudeBusDouble = 0.;
    double latitudeArretDouble = 0.;
    double longitudeArretDouble = 0.;

    double latitudeBusDoubleRadians = 0.;
    double longitudeBusDoubleRadians = 0.;
    double latitudeArretDoubleRadians = 0.;
    double longitudeArretDoubleRadians = 0.;

    // Conversion des QString en double (degrés)

    latitudeBusDouble = latitudeBus.toDouble();
    longitudeBusDouble = longitudeBus.toDouble();
    latitudeArretDouble = latitudeArret.toDouble();
    longitudeArretDouble = longitudeArret.toDouble();

    // Conversion Degrés > Radians

    latitudeBusDoubleRadians = (latitudeBusDouble*(PI/180));
    longitudeBusDoubleRadians = (longitudeBusDouble*(PI/180));
    latitudeArretDoubleRadians = (latitudeArretDouble*(PI/180));
    longitudeArretDoubleRadians = (longitudeArretDouble*(PI/180));

    // Formules de calcul avec les coordonnées du bus et de la zone d'Arret
       (latitude et longitude)

    distanceLatitude = (latitudeArretDoubleRadians - latitudeBusDoubleRadians)/
      2;
    distanceLongitude = (longitudeArretDoubleRadians -
      longitudeBusDoubleRadians)/2;
    double a = (sin(distanceLatitude)*sin(distanceLatitude))+(cos(
      latitudeBusDoubleRadians)*cos(latitudeArretDoubleRadians)*sin(distanceLongitude)*sin(
      distanceLongitude));
    distance = 2*atan2(sqrt(a), sqrt(1-a));

    return ((rayonTerreRadians*distance)/1000);
}
```

9.13.3.7 TInformationVoyageur : :chargerItineraire (QString *numItineraire*)

Paramètres

numItineraire	

Références bdd, itineraire, nombreArrets, numeroArret, pret, et BaseDeDonnees : :recuperer().

Référencé par actualiserCourse().

```
{
    itineraire.clear();

    // requete SQL
    QString requete = "SELECT itineraire.idLigne, nomCourt, destination,
       heureDepart, numeroSequence, nomLieu, latitude, longitude FROM itineraire INNER JOIN
       ligne ON ligne.idLigne=itineraire.idLigne INNER JOIN arret ON
       arret.idItineraire=itineraire.idItineraire INNER JOIN lieu ON lieu.idArret=arret.idArret WHERE
       itineraire.idItineraire='" + numItineraire + "' ORDER BY arret.numeroSequence ASC";
    bdd->recuperer(requete, itineraire);

    if(!itineraire.isEmpty())
    {
        pret = true;

        nombreArrets = itineraire.size();
        numeroArret = 0;

        #ifdef DEBUG_PANNEAU
        qDebug() << Q_FUNC_INFO << "ITINERAIRE CHARGE !" << itineraire << "
      TAILLE" << nombreArrets;
        #endif
```

```
    }
    else
    {
        pret = false;
        #ifdef DEBUG_PANNEAU
        qDebug() << Q_FUNC_INFO << "Aucun itineraire n'a été chargé !";
        #endif
    }
}
```

9.13.3.8 TInformationVoyageur : :declencherDebutArret ( )

Références setZoneArret(), et signalerArret().

Référencé par detecterArret().

```
{
    setZoneArret(true);
    signalerArret();

    // TODO : cas du terminus
}
```

9.13.3.9 TInformationVoyageur : :declencherFinArret ( )

Références arret, COURSE_TERMINEE, enCourse, nombreArrets, numeroArret, setZoneArret(), et signalerArret().

Référencé par detecterArret().

```
{
    #ifdef DEBUG_PANNEAU
    qDebug() << Q_FUNC_INFO << "Arret : " << arret;
    #endif

    if (arret == true)
    {
        setZoneArret(false);
        signalerArret();
        if(numeroArret < (nombreArrets-1))
        {
            numeroArret++; //on passe au prochain arret
        }
        else
        {
            // la course est terminée
            enCourse = COURSE_TERMINEE;
        }
    }
}
```

9.13.3.10 TInformationVoyageur : :detecterArret ( )

Références declencherDebutArret(), declencherFinArret(), detecterDebutArret(), detecterFinArret(), et numeroArret.

Référencé par main().

```
{
    bool debut = detecterDebutArret();
    bool fin = detecterFinArret();

    #ifdef DEBUG_PANNEAU
    qDebug() << Q_FUNC_INFO << "Numero arret : " << numeroArret;
    #endif

    #ifdef DEBUG_PANNEAU
    qDebug() << Q_FUNC_INFO << "Debut : " << debut;
    #endif

    if (debut == true)
    {
        declencherDebutArret();
    }

    #ifdef DEBUG_PANNEAU
    qDebug() << Q_FUNC_INFO << "Fin : " << fin;
    #endif

    if (fin == true)
    {
        declencherFinArret();
    }
}
```

9.13.3.11 TInformationVoyageur : :detecterDebutArret ( )

Renvoie

bool

Références calculerDistance(), itineraire, latitudeBus, longitudeBus, numeroArret, et rayon.

Référencé par detecterArret().

```
{
    if(latitudeBus.isEmpty() || longitudeBus.isEmpty())
    {
        return false;
    }

    QString latitudeArret, longitudeArret;

    latitudeArret = itineraire.at(numeroArret).at(6);
    longitudeArret = itineraire.at(numeroArret).at(7);

    double distance = calculerDistance(latitudeBus, longitudeBus, latitudeArret
      , longitudeArret);

    #ifdef DEBUG_PANNEAU
    qDebug() << Q_FUNC_INFO << "Latitude/Longitude BUS : " << latitudeBus <<
      longitudeBus;
    qDebug() << Q_FUNC_INFO << "Latitude/Longitude ARRET : " << latitudeArret <
      < longitudeArret;
    qDebug() << Q_FUNC_INFO << "Distance : " << distance << "Rayon : " << rayon
      ;
    #endif

    if (distance < rayon)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

9.13.3.12 TInformationVoyageur : :detecterFinArret ( )

Renvoie

bool

Références calculerDistance(), itineraire, latitudeBus, longitudeBus, numeroArret, et rayon.

Référencé par detecterArret().

```
{
    if(latitudeBus.isEmpty() || longitudeBus.isEmpty())
        return false;

    QString latitudeArret, longitudeArret;

    latitudeArret = itineraire.at(numeroArret).at(6);
    longitudeArret = itineraire.at(numeroArret).at(7);

    double distance = calculerDistance(latitudeBus, longitudeBus, latitudeArret
      , longitudeArret);

    #ifdef DEBUG_PANNEAU
    qDebug() << Q_FUNC_INFO << "Latitude/Longitude BUS : " << latitudeBus <<
      longitudeBus;
    qDebug() << Q_FUNC_INFO << "Latitude/Longitude ARRET : " << latitudeArret <
      < longitudeArret;
    qDebug() << Q_FUNC_INFO << "Distance : " << distance << "Rayon : " << rayon
      ;
    #endif

    if (distance > rayon)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

9.13.3.13 TInformationVoyageur : :estEnCourse ( )

Renvoie

int

Références EN_COURSE, enCourse, nombreArrets, numeroArret, et PAS_EN_COURSE.

Référencé par main().

```
{
    // Si le bus est sur le parcours

    if(numeroArret == -1)
    {
        enCourse = PAS_EN_COURSE;
    }
    else if(numeroArret < nombreArrets)
    {
        enCourse = EN_COURSE;
    }

    return enCourse;
}
```

9.13.3.14 TInformationVoyageur : :estPret ( )

Renvoie

bool

Références pret.

Référencé par main().

```
{
    // Un itineraire a été chargé

    return pret;
}
```

9.13.3.15 TInformationVoyageur : :finir ( ) [slot]

Références annuler(), et fini.

Référencé par main().

```
{
    fini = true;
    annuler();
}
```

9.13.3.16 TInformationVoyageur : :finirCourse ( )

Renvoie

bool

Référencé par main().

```
{
    // Terminus
    // emit terminerCourse();

    return false;
}
```

9.13.3.17 TInformationVoyageur : :lireParametres ( )

Références periode.

Référencé par main().

```
{
    // Le nom du fichier INI : nom-executable.ini
    QString fichierINI = qApp->applicationName() + ".ini"; // siv.ini

    QSettings parametres(fichierINI, QSettings::IniFormat);

    // periode course terminée
    periode = parametres.value("informationsVoyageur/periode", "10000").toInt()
      ;

    // Affichage de débugage
    #ifdef DEBUG_PANNEAU
    qDebug() << Q_FUNC_INFO << QString::fromUtf8("période : %1").arg(periode);
    #endif
}
```

9.13.3.18 TInformationVoyageur : :main ( ) [slot]

Références attendrePeriode(), COURSE_TERMINEE, detecterArret(), diffusionMessages, EN_COURSE, enCourse, estEnCourse(), estPret(), fini, TDiffusionMessages : :finir(), finirCourse(), lireParametres(), PAS_EN_COURSE, et threadDiffusionMessages.

Référencé par TInformationVoyageur().

```
{

    int enCourse = PAS_EN_COURSE;

    #ifdef DEBUG_PANNEAU
        qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
    #endif

    threadDiffusionMessages.start();

    lireParametres();

    while(!fini)
    {
        #ifdef DEBUG_PANNEAU
        qDebug() << Q_FUNC_INFO << " estPret() =" << estPret();
        #endif
        if(estPret())
        {
            enCourse = estEnCourse();
            #ifdef DEBUG_PANNEAU
            qDebug() << Q_FUNC_INFO << " enCourse() =" << enCourse;
            #endif
            if (enCourse == EN_COURSE)
            {
                detecterArret();
            }
            else if (enCourse == COURSE_TERMINEE)
            {
                finirCourse();
            }
        }
        attendrePeriode();
    }

    diffusionMessages.finir();
    threadDiffusionMessages.quit();
    threadDiffusionMessages.wait();
}
```

9.13.3.19 TInformationVoyageur : :msleep (unsigned long *sleepMS*) [inline, private]

Paramètres

sleepMS	

Références mutex, et waitCondition.

Référencé par attendrePeriode().

```
    {
```

```
        waitCondition.wait(&mutex, sleepMS);
    }
```

9.13.3.20 TInformationVoyageur : :setZoneArret (bool *etat*)

Paramètres

etat	

Références arret.

Référencé par declencherDebutArret(), et declencherFinArret().

```
{
    arret = etat;
}
```

9.13.3.21 TInformationVoyageur : :signalerArret ( )

Références actualiserInformationsVoyageur(), arret, hautParleur, itineraire, nombreArrets, numeroArret, et HautParleur : :parle().

Référencé par declencherDebutArret(), et declencherFinArret().

```
{
    QString numLigne, destination, arretEnCours, prochainArret;

    numLigne = itineraire.at(0).at(1);
    destination = itineraire.at(0).at(2);
    arretEnCours = itineraire.at(numeroArret).at(5);
    if(numeroArret < (nombreArrets-1))
    {
        prochainArret = itineraire.at(numeroArret+1).at(5);
    }
    else
    {
        prochainArret = ""; // pas de prochain arrêt
    }

    #ifdef DEBUG_PANNEAU
    qDebug() << Q_FUNC_INFO << "numLigne : " << numLigne << "destination : " <<
       destination << "arretEnCours : " << arretEnCours << "prochainArret : " <<
      prochainArret;
    #endif

    emit actualiserInformationsVoyageur(numLigne, destination, arretEnCours,
      prochainArret, numeroArret);

    if (arret == true)
    {

        hautParleur->parle(arretEnCours);
    }
    else
    {
        if(!prochainArret.isEmpty())
        {

            hautParleur->parle(prochainArret);
        }
    }
}
```

9.13.3.22 TInformationVoyageur : :terminer ( ) [slot]

Référencé par TInformationVoyageur().

```
{
    #ifdef DEBUG_PANNEAU
    qDebug() << Q_FUNC_INFO << QThread::currentThreadId() << this;
    #endif
}
```

9.13.4 Documentation des données membres

9.13.4.1 bool TInformationVoyageur : :arret [private]

Un bool permmettant de savoir si le bus est sur un arret ou non

Référencé par declencherFinArret(), setZoneArret(), signalerArret(), et TInformationVoyageur().

9.13.4.2 BaseDeDonnees* TInformationVoyageur : :bdd [private]

Un pointeur sur un objet de type BaseDeDonnees

Référencé par chargerItineraire(), et TInformationVoyageur().

9.13.4.3 TDiffusionMessages TInformationVoyageur : :diffusionMessages [private]

Objet diffusionMessages de type TDiffusionMessages

Référencé par main(), et TInformationVoyageur().

9.13.4.4 int TInformationVoyageur : :enCourse [private]

Un int qui permet de savoir si le bus est en course ou non

Référencé par declencherFinArret(), estEnCourse(), main(), et TInformationVoyageur().

9.13.4.5 bool TInformationVoyageur : :fini [private]

Un bool permmettant de savoir lorsque le thread est terminé

Référencé par finir(), main(), et TInformationVoyageur().

9.13.4.6 HautParleur* TInformationVoyageur : :hautParleur [private]

Objet hautParleur de type HautParleur

Référencé par signalerArret(), TInformationVoyageur(), et ~TInformationVoyageur().

9.13.4.7 QVector<QStringList> TInformationVoyageur : :itineraire [private]

Objet itineraire de type QVector<QStringList>

Référencé par chargerItineraire(), detecterDebutArret(), detecterFinArret(), et signalerArret().

9.13.4.8 QString TInformationVoyageur : :latitudeBus [private]

Un QString permettant de gérer la latitude GPS du bus

Référencé par actualiserLocalisation(), detecterDebutArret(), et detecterFinArret().

9.13.4.9 QString TInformationVoyageur : :longitudeBus [private]

un QString permettant de gérer la longitude GPS du bus

Référencé par actualiserLocalisation(), detecterDebutArret(), et detecterFinArret().

9.13.4.10 QMutex TInformationVoyageur : :mutex [private]

mutex pour la gestion des temporisations

Référencé par msleep().

9.13.4.11 int TInformationVoyageur : :nombreArrets [private]

Un int permettant de gérer le nombre d'arrêts sur une course

Référencé par chargerItineraire(), declencherFinArret(), estEnCourse(), signalerArret(), et TInformationVoyageur().

9.13.4.12 int TInformationVoyageur : :numeroArret [private]

Un int permettant de gérer le séquencement des arrêts sur une course

Référencé par chargerItineraire(), declencherFinArret(), detecterArret(), detecterDebutArret(), detecterFinArret(), estEnCourse(), signalerArret(), et TInformationVoyageur().

9.13.4.13 int TInformationVoyageur : :periode [private]

Un int correspondant au temps d'attente avant de redetecter si le bus est en course ou non

Référencé par attendrePeriode(), lireParametres(), et TInformationVoyageur().

9.13.4.14 bool TInformationVoyageur : :pret [private]

Un bool permettant de savoir si oui ou non le bus est pret (en fonction du conducteur)

Référencé par chargerItineraire(), estPret(), et TInformationVoyageur().

9.13.4.15 double TInformationVoyageur : :rayon [private]

Un double correspondant au rayon de detection des arrêts

Référencé par detecterDebutArret(), detecterFinArret(), et TInformationVoyageur().

9.13.4.16 QThread TInformationVoyageur : :threadDiffusionMessages [private]

Objet threadDiffusionMessages de type QThread

Référencé par main(), et TInformationVoyageur().

9.13.4.17 QWaitCondition TInformationVoyageur : :waitCondition [private]

pour la gestion des temporisations

Référencé par annuler(), et msleep().

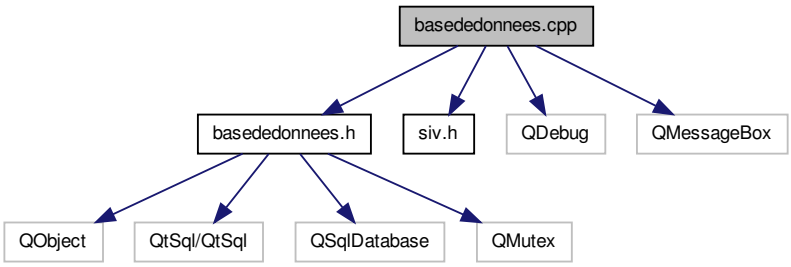
La documentation de cette classe a été générée à partir des fichiers suivants :
– tinformationvoyageur.h
– tinformationvoyageur.cpp

10 Documentation des fichiers

10.1 Référence du fichier basededonnees.cpp

```
#include "basededonnees.h" #include "siv.h" #include <QDebug> #include <QMessageBox> ×
```

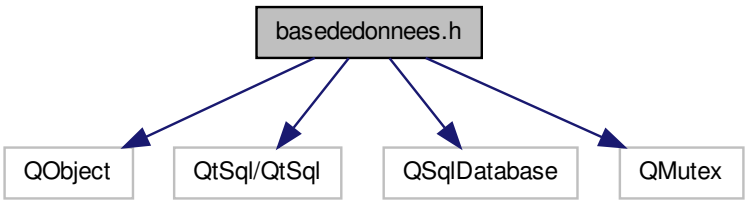
Graphe des dépendances par inclusion de basededonnees.cpp :



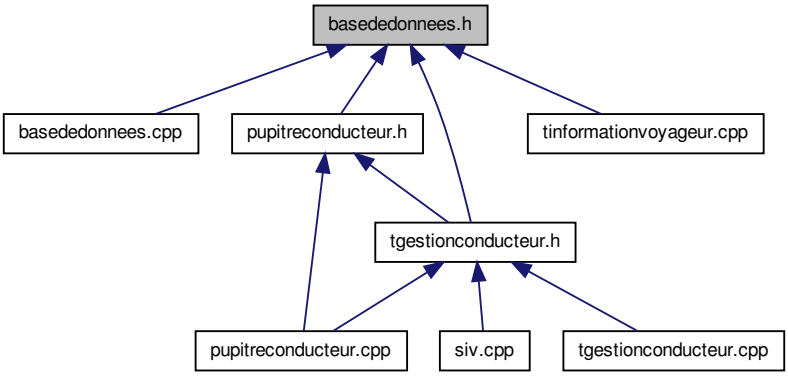
10.2 Référence du fichier basededonnees.h

```
#include <QObject> #include <QtSql/QtSql> #include <QSqlDatabase> #include <QMutex> ×
```

Graphe des dépendances par inclusion de basededonnees.h :



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



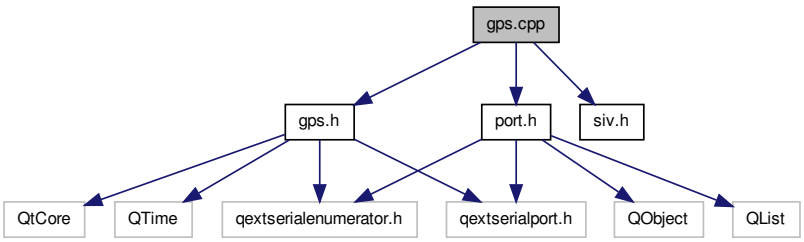
Classes

– class BaseDeDonnees
Classe de gestion de base de données SQLite.

10.3 Référence du fichier Changelog.dox

10.4 Référence du fichier gps.cpp

`#include "gps.h"` `#include "port.h"` `#include "siv.h"` Graphe des dépendances par inclusion de gps.cpp :



Macros

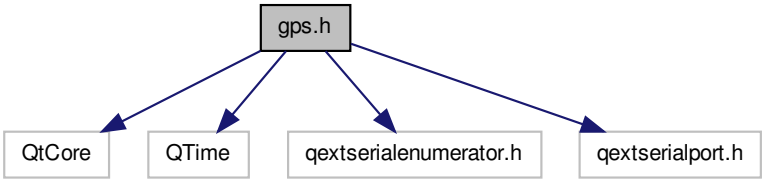
– #define PORT_DEFAUT "/dev/ttyUSB2"

10.4.1 Documentation des macros

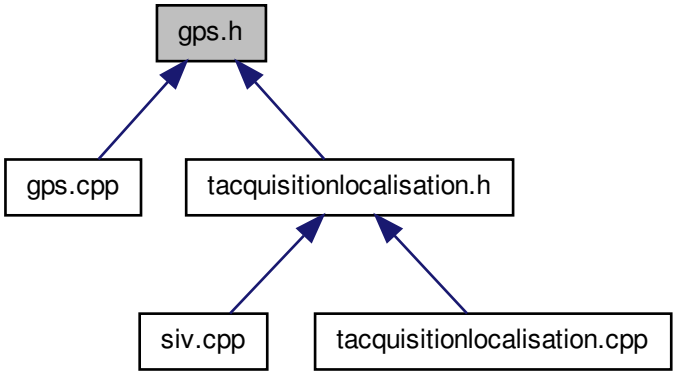
10.4.1.1 #define PORT_DEFAUT "/dev/ttyUSB2"

10.5 Référence du fichier gps.h

`#include <QtCore> #include <QTime> #include "qextserialenumerator.h" #include "qextserialport h"` Graphe des dépendances par inclusion de gps.h :



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



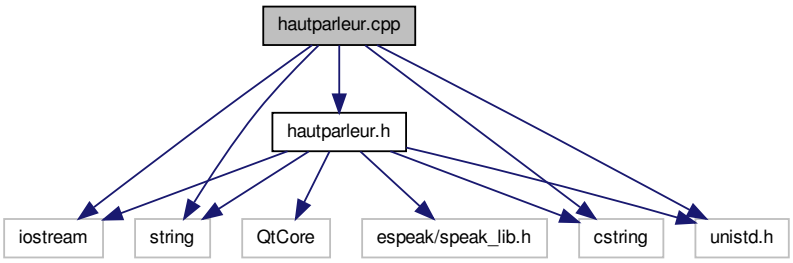
Classes

– class GPS
Classe qui permet l'aquisition et le traitement des données de géolocalisation.

10.6 Référence du fichier hautparleur.cpp

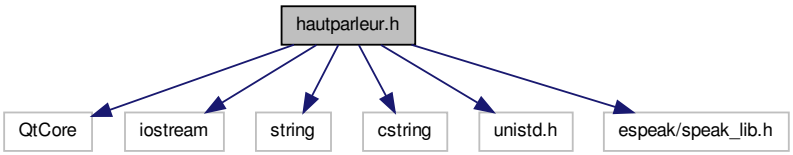
`#include "hautparleur.h" #include <iostream> #include <string> #include <cstring>` ×

`#include <unistd.h>` Graphe des dépendances par inclusion de hautparleur.cpp :

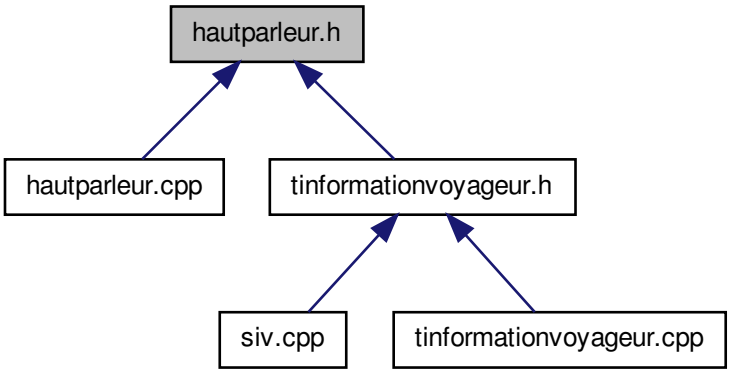


10.7 Référence du fichier hautparleur.h

`#include <QtCore> #include <iostream> #include <string> #include <cstring> #include <unistd.h> #include <espeak/speak_lib.h>` Graphe des dépendances par inclusion de hautparleur.h :



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

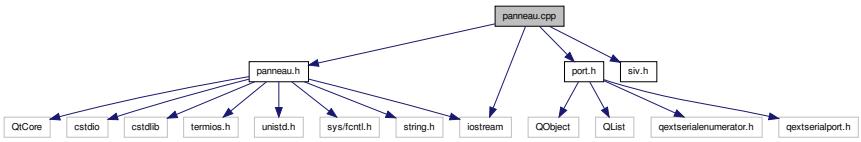


Classes

– class HautParleur
Classe qui permet de générer des messages sonores.

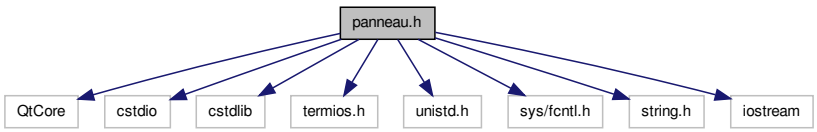
10.8 Référence du fichier panneau.cpp

`#include "panneau.h"` `#include "port.h"` `#include "siv.h"` `#include <iostream>` Graphe des dépendances par inclusion de panneau.cpp :

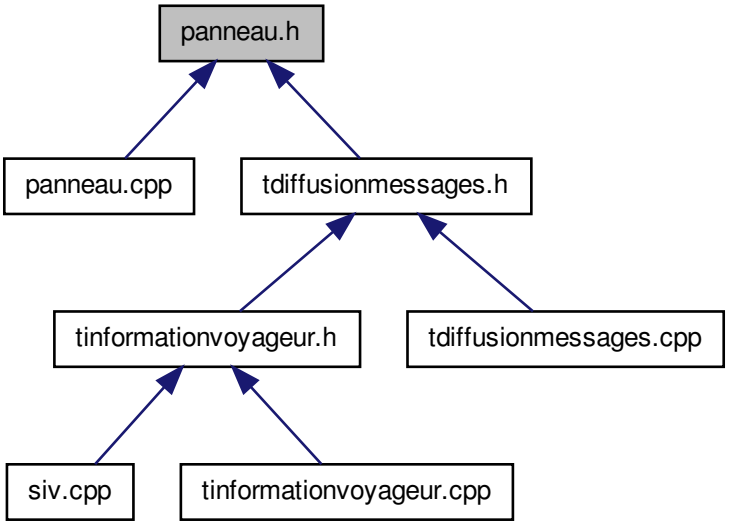


10.9 Référence du fichier panneau.h

`#include <QtCore>` `#include <cstdio>` `#include <cstdlib>` `#include <termios.h>` `#include <unistd.h>` `#include <sys/fcntl.h>` `#include <string.h>` `#include <iostream>` Graphe des dépendances par inclusion de panneau.h :



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



Classes

– class Panneau
Classe qui gère l'affichage d'informations sur le panneau.

Macros

- #define PORT_DEFAUT "/dev/panneau"
  Nom et chemin vers le fichier de périphérique du panneau.
- #define MAX_TRAME 64
  Longueur maximale d'une trame.
- #define MAX_TEXTE 10
  Longueur maximale d'affichage d'un texte.
- #define SOT 0x01
  Début de trame (Start Of Transmission)
- #define NUM 0x30
  Numéro de panneau.
- #define CMD 0x06
  Commande d'affichage d'un texte.
- #define AUX 0x0F
  Fonction auxiliare.
- #define F 0x46
  Le caractère 'F'.
- #define Z 0x30
  Le caractère Zéro.
- #define STX 0x02
  Début de texte (Start of TeXte)
- #define CHAN 0x31
  Numéro de canal de l'afficheur.
- #define LINE 0x32
  Numéro de ligne de l'afficheur.
- #define POS_DEFAUT 11
  Position par défaut pour le premier caractère du texte.
- #define ETX 0x03
  Fin de texte (End of TeXte)
- #define EOT 0x04
  Fin de trame (Enf Of Transmission)
- #define FIN 0x00
  Fin de chaîne (caractère NUL)

10.9.1 Documentation des macros

10.9.1.1 #define AUX 0x0F

Référencé par Panneau : :fabriquerTrame().

10.9.1.2 #define CHAN 0x31

Référencé par Panneau : :fabriquerTrame().

10.9.1.3 #define CMD 0x06

Référencé par Panneau : :fabriquerTrame().

10.9.1.4 #define EOT 0x04

Référencé par Panneau : :fabriquerTrame().

10.9.1.5 #define ETX 0x03

Référencé par Panneau : :fabriquerTrame().

10.9.1.6 #define F 0x46

Référencé par Panneau : :fabriquerTrame().

10.9.1.7 #define FIN 0x00

Référencé par Panneau : :afficher(), Panneau : :defiler(), Panneau : :effacer(), et Panneau : :fabriquerTrame().

10.9.1.8 #define LINE 0x32

Référencé par Panneau : :fabriquerTrame().

10.9.1.9 #define MAX_TEXTE 10

Référencé par TDiffusionMessages : :afficherArret(), TDiffusionMessages : :afficherDestination(), TDiffusionMessages : :afficher-Ligne(), et Panneau : :defiler().

10.9.1.10 #define MAX_TRAME 64

Référencé par Panneau : :afficher(), Panneau : :defiler(), Panneau : :effacer(), et Panneau : :fabriquerTrame().

10.9.1.11 #define NUM 0x30

Référencé par Panneau : :fabriquerTrame().

10.9.1.12 #define PORT_DEFAUT "/dev/panneau"

10.9.1.13 #define POS_DEFAUT 11

10.9.1.14 #define SOT 0x01

Référencé par Panneau : :fabriquerTrame().

10.9.1.15 #define STX 0x02

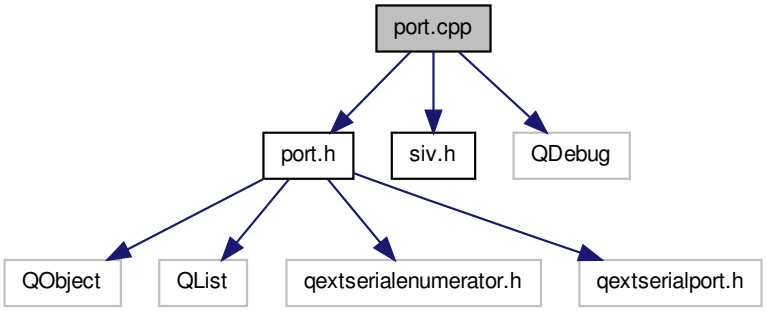
Référencé par Panneau : :fabriquerTrame().

10.9.1.16 #define Z 0x30

Référencé par Panneau : :fabriquerTrame().

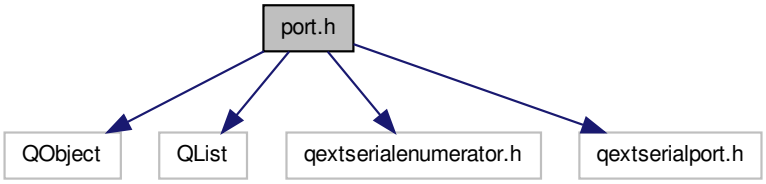
10.10 Référence du fichier port.cpp

`#include "port.h"` `#include "siv.h"` `#include <QDebug>` Graphe des dépendances par inclusion de port.-cpp :

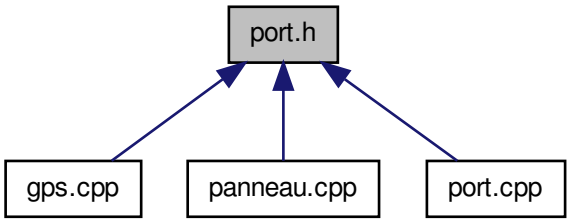


10.11 Référence du fichier port.h

`#include <QObject>` `#include <QList>` `#include "qextserialenumerator.h"` `#include "qextserialport.h"` Graphe des dépendances par inclusion de port.h :



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



Classes

– class Port
Classe de gestion d'un port série.

Macros

– #define BUFFERSIZE 1024

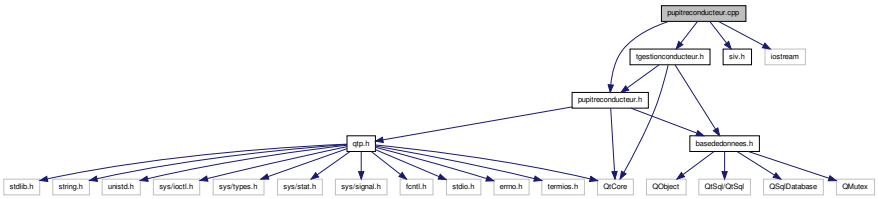
10.11.1 Documentation des macros

10.11.1.1 #define BUFFERSIZE 1024

Référencé par Port : :recevoir().

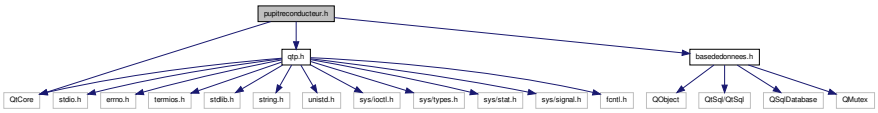
10.12 Référence du fichier pupitreconducteur.cpp

`#include "pupitreconducteur.h"` `#include "tgestionconducteur.h"` `#include "siv.h"` `#include <iostream>` Graphe des dépendances par inclusion de pupitreconducteur.cpp :

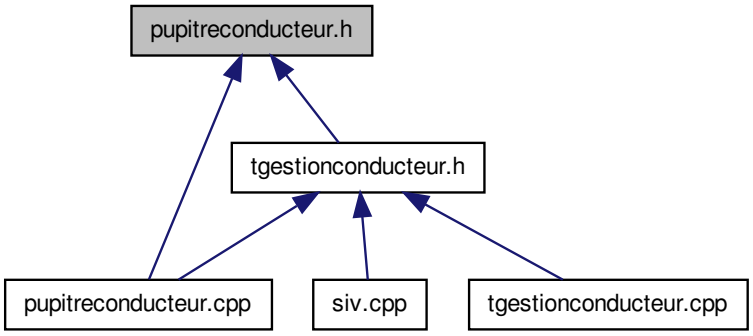


10.13 Référence du fichier pupitreconducteur.h

`#include <QtCore>` `#include "qtp.h"` `#include "basededonnees.h"` Graphe des dépendances par inclusion de pupitreconducteur.h :



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



Classes

– class PupitreConducteur
Classe qui gère le pupitre du conducteur.

Macros

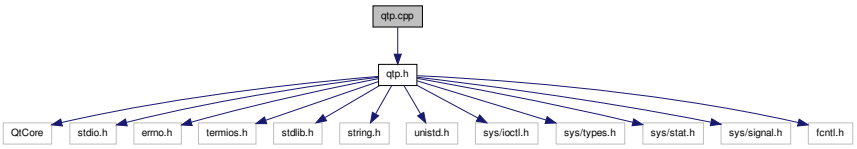
– #define LG_CODE 4

10.13.1 Documentation des macros

10.13.1.1 #define LG_CODE 4

10.14 Référence du fichier qtp.cpp

`#include "qtp.h"` Graphe des dépendances par inclusion de qtp.cpp :



Variables

– static char cmds [][LG_CMD]

10.14.1 Documentation des variables

10.14.1.1 char cmds[][LG_CMD] [static]

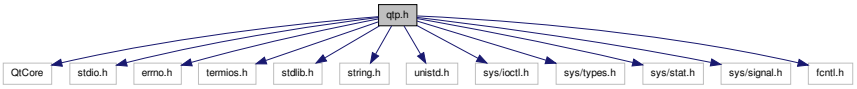
Valeur initiale :

```
{
                                {0x1B, 0x59, 0x00, 0x00, 0x00, 4},
                                {0x1B, 0x4B, 0x00, 0x00, 0x00, 2},
                                {0x1B, 0x6B, 0x00, 0x00, 0x00, 2},
                                {0x1B, 0x50, 0x00, 0x00, 0x00, 2},
                                {0x1B, 0x4F, 0x00, 0x00, 0x00, 2},
                                {0x1B, 0x51, 0x00, 0x00, 0x00, 2},
                                {0x1B, 0x33, 0x00, 0x00, 0x00, 2},
                                {0x1B, 0x21, 0x4E, 0x00, 0x00, 4},
                                {0x1B, 0x21, 0x6E, 0x00, 0x00, 3},
                                {0x1B, 0x37, 0x00, 0x00, 0x00, 4},
                                {0x1B, 0x35, 0x00, 0x00, 0x00, 2},
                                {0x1B, 0x36, 0x00, 0x00, 0x00, 2},
                                {0x1B, 0x21, 0x35, 0x00, 0x00, 3},
                                {0x1B, 0x21, 0x36, 0x00, 0x00, 3},
                                {0x1B, 0x56, 0x00, 0x00, 0x00, 2},
                                {0x1B, 0x6E, 0x00, 0x00, 0x00, 2},
                                {0x1B, 0x21, 0x43, 0x00, 0x00, 4},
                                {0x1B, 0x21, 0x45, 0x00, 0x00, 4},
                                {0x1B, 0x21, 0x44, 0x00, 0x00, 5},
                                {0x1B, 0x21, 0x45, 0x00, 0x00, 5},
                                {0x1B, 0x21, 0x49, 0x00, 0x00, 4},
                                {0x1B, 0x49, 0x00, 0x00, 0x00, 2}
                               }
```

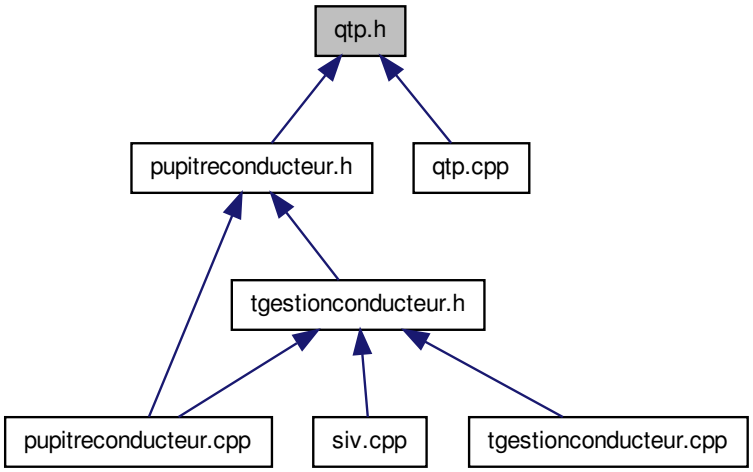
Référencé par QTP : :clearEndofline(), QTP : :clearEndofpage(), QTP : :cursorBlink(), QTP : :cursorOff(), QTP : :cursorOn(), QTP : :gotoxy(), et QTP : :requestReady().

10.15 Référence du fichier qtp.h

`#include <QtCore> #include <stdio.h> #include <errno.h> #include <termios.h> #include <stdlib.h> #include <string.h> #include <unistd.h> #include <sys/ioctl.h> #include <sys/types.h> #include <sys/stat.h> #include <sys/signal.h> #include <fcntl.h>` Graphe des dépendances par inclusion de qtp.h :



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



Classes

– class QTP
Classe de bas niveau qui gère le terminal QTP.

Macros

– #define QTP_READY 0x06
– #define QTP_NOTREADY 0x15
– #define MAX_MESS_256 9
– #define MAX_MESS_512 22
– #define MAX_MESS_1024 47
– #define MAX_MESS_2048 99
– #define MAX_LENGTH 20
– #define NB_CMDS 21
– #define LG_CMD 6
– #define ABS 0
– #define CLEAR_ENDOFLINE 1
– #define CLEAR_ENDOFPAGE 2
– #define CURSOR_OFF 3
– #define CURSOR_ON 4
– #define BLINK 5
– #define REQUEST_READY 6
– #define TEST_WRITE 7
– #define TEST_READ 8
– #define KEY_RECONFIG 9
– #define KEYCLICK_ON 10
– #define KEYCLICK_OFF 11
– #define KEYCLICK_ON_SAVE 12
– #define KEYCLICK_OFF_SAVE 13
– #define READ_VERSION 14
– #define READ_NUMBER 15
– #define MESSAGE_STORING 16
– #define MESSAGE_READING 17
– #define MESSAGE_DISPLAY 18
– #define MESSAGE_SCROLL 19
– #define INPUTS_CONFIG 20
– #define INPUTS_READING 21
– #define CURSOR_LEFT 0x15
– #define CURSOR_RIGHT 0x06
– #define CURSOR_DOWN 0x0A
– #define CURSOR_UP 0x1A
– #define CURSOR_HOME 0x01
– #define CR 0x0D
– #define LF 0x0A

- #define CRLF 0x1D
- #define BS 0x08
- #define CLEAR_PAGE 0x0C
- #define CLEAR_LINE 0x19
- #define BELL 0x07
- #define BEEP 0x07
- #define ESC 0x1B
- #define ACK 0x06
- #define NACK 0x15
- #define TEST 0x55
- #define ERROR -1
- #define VRAI 1
- #define FAUX 0
- #define LG_LIGNE 20
- #define NB_LIGNES 4
- #define QTP_ECHO 1
- #define QTP_NOECHO 0
- #define DIESE 0x0D
- #define ETOILE 0x1B
- #define NB_CHAR(x) (cmds[x][LG_CMD-1])

10.15.1 Documentation des macros

10.15.1.1 #define ABS 0

Référencé par QTP : :cursorBlink(), et QTP : :gotoxy().

10.15.1.2 #define ACK 0x06

10.15.1.3 #define BEEP 0x07

10.15.1.4 #define BELL 0x07

10.15.1.5 #define BLINK 5

Référencé par QTP : :cursorBlink().

10.15.1.6 #define BS 0x08

10.15.1.7 #define CLEAR_ENDOFLINE 1

Référencé par QTP : :clearEndofline().

10.15.1.8 #define CLEAR_ENDOFPAGE 2

10.15.1.9 #define CLEAR_LINE 0x19

10.15.1.10 #define CLEAR_PAGE 0x0C

Référencé par QTP : :clearEndofpage().

10.15.1.11 #define CR 0x0D

Référencé par QTP : :lireTouche(), QTP : :recv(), et PupitreConducteur : :saisirCodeConducteur().

10.15.1.12 #define CRLF 0x1D

10.15.1.13 #define CURSOR_DOWN 0x0A

Référencé par QTP : :lireTouche().

10.15.1.14 #define CURSOR_HOME 0x01

10.15.1.15 #define CURSOR_LEFT 0x15

10.15.1.16 #define CURSOR_OFF 3

Référencé par QTP : :cursorOff().

10.15.1.17 #define CURSOR_ON 4

Référencé par QTP : :cursorOn().

10.15.1.18 #define CURSOR_RIGHT 0x06

10.15.1.19 #define CURSOR_UP 0x1A

Référencé par QTP : :lireTouche().

10.15.1.20 #define DIESE 0x0D

Référencé par PupitreConducteur : :lireTouche().

10.15.1.21 #define ERROR -1

Référencé par QTP : :recv(), et QTP : :send().

10.15.1.22 #define ESC 0x1B

Référencé par QTP : :lireTouche(), et PupitreConducteur : :saisirCodeConducteur().

10.15.1.23 #define ETOILE 0x1B

Référencé par PupitreConducteur : :lireTouche().

10.15.1.24 #define FAUX 0

Référencé par QTP : :recv().

10.15.1.25 #define INPUTS_CONFIG 20

10.15.1.26 #define INPUTS_READING 21

10.15.1.27 #define KEY_RECONFIG 9

10.15.1.28 #define KEYCLICK_OFF 11

10.15.1.29 #define KEYCLICK_OFF_SAVE 13

10.15.1.30 #define KEYCLICK_ON 10

10.15.1.31 #define KEYCLICK_ON_SAVE 12

10.15.1.32 #define LF 0x0A

Référencé par QTP : :recv().

10.15.1.33 #define LG_CMD 6

10.15.1.34 #define LG_LIGNE 20

Référencé par PupitreConducteur : :afficherAucunService(), PupitreConducteur : :afficherCodeBon(), PupitreConducteur-: :afficherCodeFaux(), PupitreConducteur : :afficherCourseDemarrer(), PupitreConducteur : :afficherCourseIndisponible(), Pupitre-Conducteur : :afficherFinService(), PupitreConducteur : :afficherMenuPrincipal(), PupitreConducteur : :saisirCodeConducteur(), et PupitreConducteur : :visualiserInformationsArret().

10.15.1.35 #define MAX_LENGTH 20

Référencé par QTP : :recv().

10.15.1.36 #define MAX_MESS_1024 47

10.15.1.37 #define MAX_MESS_2048 99

Référencé par QTP : :recv().

10.15.1.38 #define MAX_MESS_256 9

10.15.1.39 #define MAX_MESS_512 22

10.15.1.40 #define MESSAGE_DISPLAY 18

10.15.1.41 #define MESSAGE_READING 17

10.15.1.42 #define MESSAGE_SCROLL 19

10.15.1.43 #define MESSAGE_STORING 16

10.15.1.44 #define NACK 0x15

10.15.1.45 #define NB_CHAR(*x*) (cmds[x][LG_CMD-1])

Référencé par QTP : :clearEndofline(), QTP : :clearEndofpage(), QTP : :cursorBlink(), QTP : :cursorOff(), QTP : :cursorOn(), QTP : :gotoxy(), et QTP : :requestReady().

10.15.1.46 #define NB_CMDS 21

10.15.1.47 #define NB_LIGNES 4

Référencé par PupitreConducteur : :afficherAucunService(), PupitreConducteur : :afficherCodeBon(), PupitreConducteur-: :afficherCodeFaux(), PupitreConducteur : :afficherCourseDemarrer(), PupitreConducteur : :afficherCourseIndisponible(), -PupitreConducteur : :afficherFinService(), PupitreConducteur : :afficherMenuPrincipal(), PupitreConducteur : :afficherMessage(), PupitreConducteur : :effacer(), PupitreConducteur : :saisirCodeConducteur(), et PupitreConducteur : :visualiserInformations-Arret().

10.15.1.48 #define QTP_ECHO 1

Référencé par QTP : :lireTouche().

10.15.1.49 #define QTP_NOECHO 0

Référencé par PupitreConducteur : :gererMenuService(), et PupitreConducteur : :saisirCodeConducteur().

10.15.1.50 #define QTP_NOTREADY 0x15

10.15.1.51 #define QTP_READY 0x06

10.15.1.52 #define READ_NUMBER 15

10.15.1.53 #define READ_VERSION 14

10.15.1.54 #define REQUEST_READY 6

Référencé par QTP : :requestReady().

10.15.1.55 #define TEST 0x55

10.15.1.56 #define TEST_READ 8

10.15.1.57 #define TEST_WRITE 7

10.15.1.58 #define VRAI 1

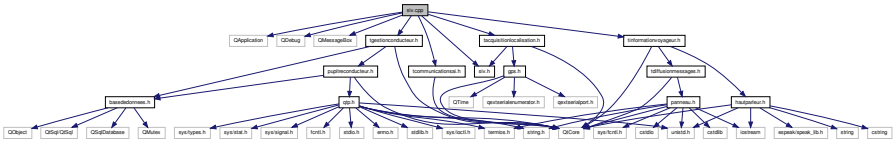
Référencé par QTP : :recv().

10.16 Référence du fichier README.dox

10.17 Référence du fichier siv.cpp

Programme principal.

`#include <QApplication>` `#include <QDebug>` `#include <QMessageBox>` `#include "siv.h"` `#include "tgestionconducteur.h"` `#include "tinformationvoyageur.h"` `#include "tacquisitionlocalisation.h"` `#include "tcommunicationsai.h"` Graphe des dépendances par inclusion de siv.cpp :



Fonctions

– int main (int argc, char **argv)
Programme principal.

10.17.1 Description détaillée

démarre les threads de l'application

10.17.2 Documentation des fonctions

10.17.2.1 main (int *argc,* char ** *argv*)

Paramètres

argc	
argv	

Renvoie

int

Références TCommunicationSAI : :finir(), TAcquisitionLocalisation : :finir(), TGestionConducteur : :finir(), et TInformationVoyageur- : :finir().

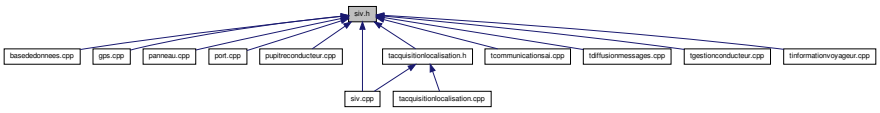
```
{
   QApplication a(argc, argv);

   a.setApplicationName("siv");

   //qDebug() << "PID : " << (int) a.applicationPid();
   qDebug() << Q_FUNC_INFO << "SIV DEBUT" << QApplication::instance()->thread()
      ->currentThreadId() << a.thread();

   TGestionConducteur gestionConducteur;
   QThread threadGestionConducteur;
   TInformationVoyageur informationVoyageur;
   QThread threadInformationVoyageur;
   TAcquisitionLocalisation acquisitionLocalisation;
   QThread threadacquisitionLocalisation;
   TCommunicationSAI communicationSAI;
   QThread threadCommunicationSAI;

   gestionConducteur.moveToThread(&threadGestionConducteur);
   informationVoyageur.moveToThread(&threadInformationVoyageur);
   acquisitionLocalisation.moveToThread(&threadacquisitionLocalisation);
   communicationSAI.moveToThread(&threadCommunicationSAI);

   QObject::connect(&threadGestionConducteur, SIGNAL(started()), &
      gestionConducteur, SLOT(main()));
   QObject::connect(&threadGestionConducteur, SIGNAL(finished()), &
      gestionConducteur, SLOT(terminer()));
   QObject::connect(&threadInformationVoyageur, SIGNAL(started()), &
      informationVoyageur, SLOT(main()));
   QObject::connect(&threadInformationVoyageur, SIGNAL(finished()), &
      informationVoyageur, SLOT(terminer()));
   QObject::connect(&threadacquisitionLocalisation, SIGNAL(started()), &
      acquisitionLocalisation, SLOT(main()));
   QObject::connect(&threadacquisitionLocalisation, SIGNAL(finished()), &
      acquisitionLocalisation, SLOT(terminer()));
   QObject::connect(&threadCommunicationSAI, SIGNAL(started()), &
      communicationSAI, SLOT(main()));
```

```
    QObject::connect(&threadCommunicationSAI, SIGNAL(finished()), &
        communicationSAI, SLOT(terminer()));

    // signaux/slots entre threads de l'application
    QObject::connect(&gestionConducteur, SIGNAL(actualiserCourse(QString)), &
        informationVoyageur, SLOT(actualiserCourse(QString)), Qt::DirectConnection);
    QObject::connect(&acquisitionLocalisation, SIGNAL(actualiserLocalisation(
        QString,QString)), &informationVoyageur, SLOT(actualiserLocalisation(QString,
        QString)), Qt::DirectConnection);
    QObject::connect(&gestionConducteur, SIGNAL(actualiserCourse(QString)), &
        acquisitionLocalisation, SLOT(demarrer(QString)), Qt::DirectConnection);
    QObject::connect(&gestionConducteur, SIGNAL(courseFinie(QString)), &
        acquisitionLocalisation, SLOT(arreterCourse(QString)), Qt::DirectConnection);
    QObject::connect(&informationVoyageur, SIGNAL(actualiserInformationsVoyageur
        (QString,QString,QString,QString,int)), &gestionConducteur, SLOT(
        actualiserInformationsVoyageur(QString,QString,QString,QString,int)), Qt::DirectConnection);
    #ifdef SIMULATION_ET2
    QObject::connect(&gestionConducteur, SIGNAL(actualiserCourse(QString)), &
        acquisitionLocalisation, SLOT(demarrerSimulationEt2(QString)), Qt::DirectConnection
        );
    #endif

    // démarre les threads
    threadGestionConducteur.start();
    threadInformationVoyageur.start();
    threadacquisitionLocalisation.start();
    threadCommunicationSAI.start();

    QMessageBox::information(0, QObject::tr("SIV"), QObject::tr("Cliquez sur Ok
         pour terminer l'application"));

    // met fin aux threads
    gestionConducteur.finir();
    threadGestionConducteur.quit();
    acquisitionLocalisation.finir();
    threadacquisitionLocalisation.quit();
    communicationSAI.finir();
    threadCommunicationSAI.quit();
    informationVoyageur.finir();
    threadInformationVoyageur.quit();

    // attend la fin des threads
    threadGestionConducteur.wait();
    threadInformationVoyageur.wait();
    threadacquisitionLocalisation.wait();
    threadCommunicationSAI.wait();

    qDebug() << Q_FUNC_INFO << "SIV FIN" << QApplication::instance()->thread()->
        currentThreadId() << a.thread();

    return 0;
}
```

10.18 Référence du fichier siv.h

Constantes globales à l'application siv.

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

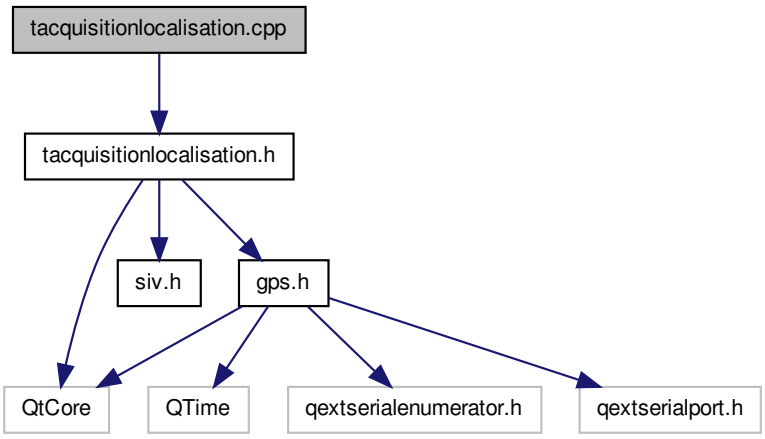


10.18.1 Description détaillée

gestion DEBUG des différentes classes

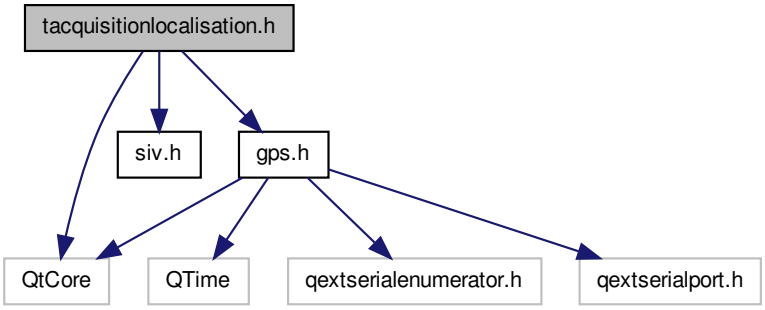
10.19 Référence du fichier tacquisitionlocalisation.cpp

`#include "tacquisitionlocalisation.h"` Graphe des dépendances par inclusion de tacquisitionlocalisation.cpp :

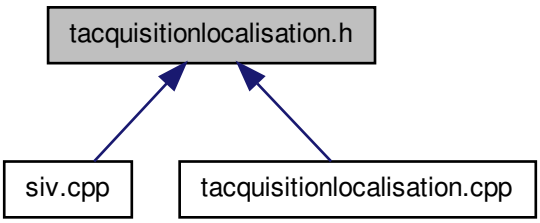


10.20 Référence du fichier tacquisitionlocalisation.h

`#include <QtCore>` `#include "siv.h"` `#include "gps.h"` Graphe des dépendances par inclusion de tacquisitionlocalisation.h :



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

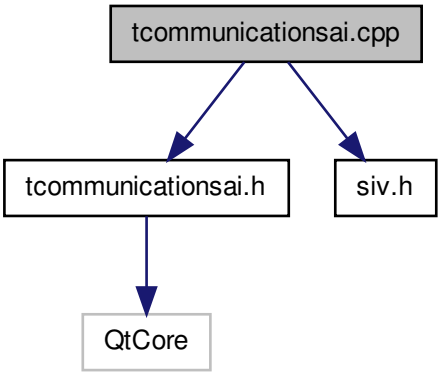


Classes

– class TAcquisitionLocalisation
Classe qui permet d'acquérir la localisation et de la diffuser.

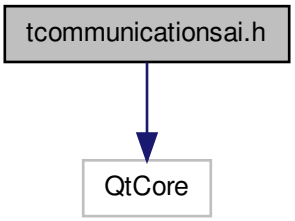
10.21 Référence du fichier tcommunicationsai.cpp

`#include "tcommunicationsai.h"` `#include "siv.h"` Graphe des dépendances par inclusion de tcommunicationsai.-cpp :

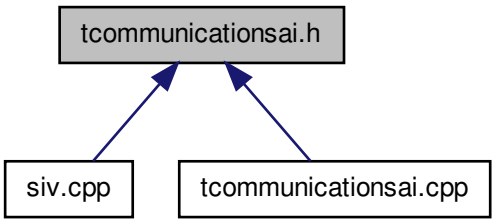


10.22 Référence du fichier tcommunicationsai.h

`#include <QtCore>` Graphe des dépendances par inclusion de tcommunicationsai.h :



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



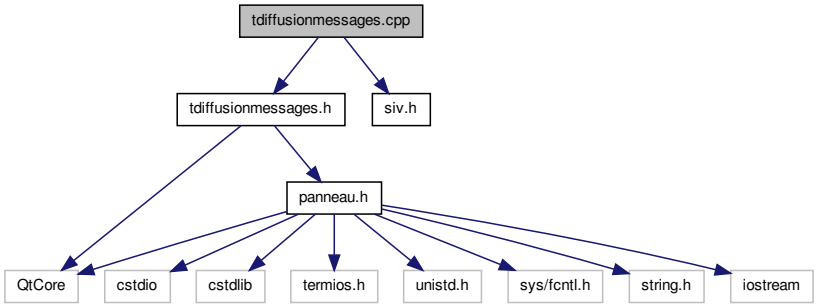
Classes

– class TCommunicationSAI
Classe qui permet de communiquer avec le SAI.

10.23 Référence du fichier tdiffusionmessages.cpp

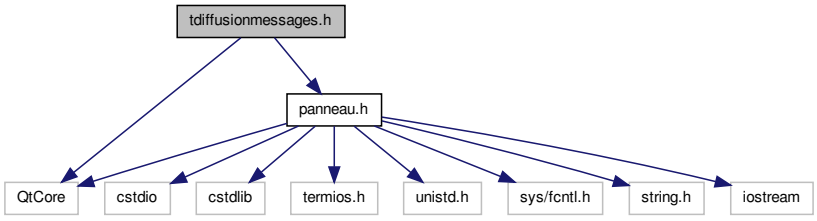
`#include "tdiffusionmessages.h"` `#include "siv.h"` Graphe des dépendances par inclusion de tdiffusionmessages.-

cpp :

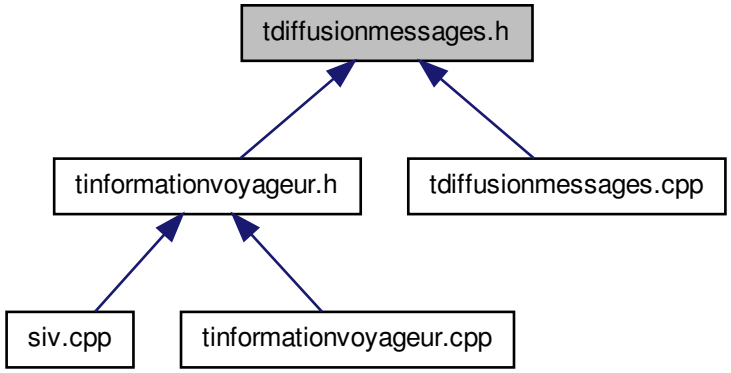


10.24 Référence du fichier tdiffusionmessages.h

`#include <QtCore>` `#include "panneau.h"` Graphe des dépendances par inclusion de tdiffusionmessages.h :



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

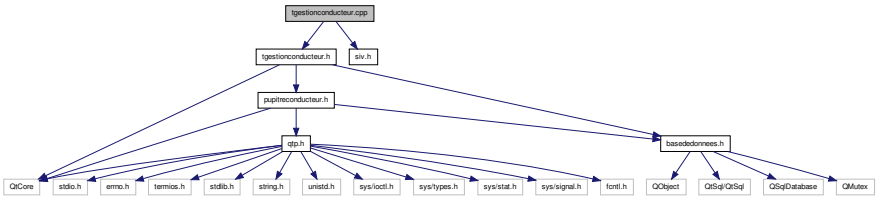


Classes

– class TDiffusionMessages
Classe qui gère la diffusion des messages informatifs dans le bus.

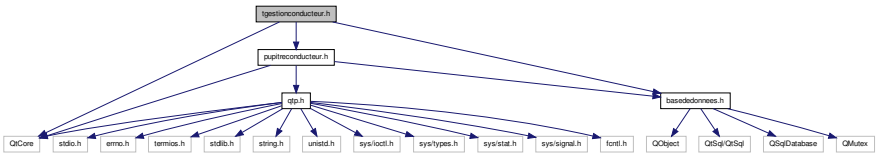
10.25 Référence du fichier tgestionconducteur.cpp

`#include "tgestionconducteur.h"` `#include "siv.h"` Graphe des dépendances par inclusion de tgestionconducteur.cpp :

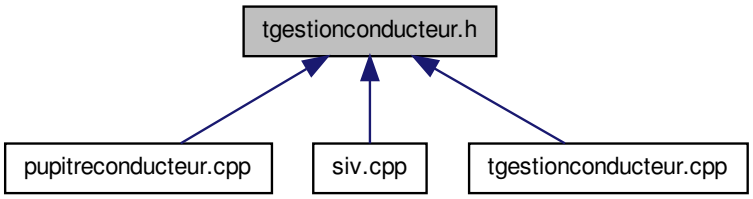


10.26 Référence du fichier tgestionconducteur.h

`#include <QtCore>` `#include "pupitreconducteur.h"` `#include "basededonnees.h"` Graphe des dépendances par inclusion de tgestionconducteur.h :



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



Classes

- struct InformationsConducteur
- class TGestionConducteur

  Classe qui gère la gestion du conducteur : Prise de service, prendre une courses, afficher l'itinéraire..

Macros

- #define AUCUN 0

  Choix du menu SERVICE.
- #define PRISE_SERVICE 1
- #define FIN_SERVICE 2
- #define DEMARRER_COURSE 3
- #define AFFICHER_COURSE 4
- #define PAS_EN_COURSE 0

  Etat du bus : PAS_EN_COURSE pour l'attribut enCourse.
- #define EN_COURSE 1

Etat du bus : EN_COURSE pour l'attribut enCourse.
- #define COURSE_TERMINEE 2
Etat du bus : COURSE_TERMINEE pour l'attribut enCourse.
- #define PAS_EN_SERVICE 0
- #define EN_SERVICE 1

10.26.1 Documentation des macros

10.26.1.1 #define AFFICHER_COURSE 4

Référencé par PupitreConducteur : :gererMenuService(), et TGestionConducteur : :main().

10.26.1.2 #define AUCUN 0

Référencé par PupitreConducteur : :gererMenuService().

10.26.1.3 #define COURSE_TERMINEE 2

Référencé par TInformationVoyageur : :declencherFinArret(), et TInformationVoyageur : :main().

10.26.1.4 #define DEMARRER_COURSE 3

Référencé par PupitreConducteur : :gererMenuService(), et TGestionConducteur : :main().

10.26.1.5 #define EN_COURSE 1

Référencé par TInformationVoyageur : :estEnCourse(), TGestionConducteur : :main(), et TInformationVoyageur : :main().

10.26.1.6 #define EN_SERVICE 1

Référencé par PupitreConducteur : :gererMenuService(), et TGestionConducteur : :main().

10.26.1.7 #define FIN_SERVICE 2

Référencé par PupitreConducteur : :gererMenuService(), et TGestionConducteur : :main().

10.26.1.8 #define PAS_EN_COURSE 0

Référencé par TGestionConducteur : :actualiserInformationsVoyageur(), TInformationVoyageur : :estEnCourse(), PupitreConducteur : :gererMenuService(), TGestionConducteur : :main(), TInformationVoyageur : :main(), et TInformationVoyageur : :TInformationVoyageur().

10.26.1.9 #define PAS_EN_SERVICE 0

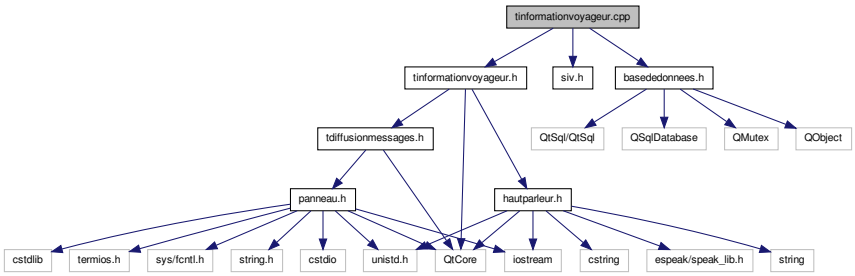
Référencé par PupitreConducteur : :gererMenuService(), et TGestionConducteur : :main().

10.26.1.10 #define PRISE_SERVICE 1

Référencé par PupitreConducteur : :gererMenuService(), et TGestionConducteur : :main().

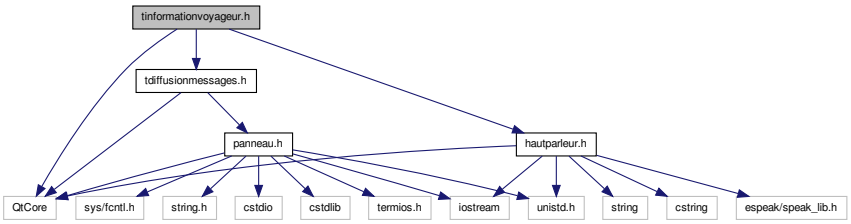
10.27 Référence du fichier tinformationvoyageur.cpp

`#include "tinformationvoyageur.h" #include "siv.h" #include "basededonnees.h"` Graphe des dépendances par inclusion de tinformationvoyageur.cpp :

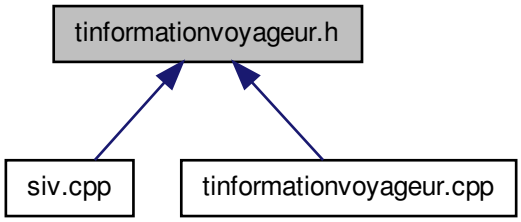


10.28 Référence du fichier tinformationvoyageur.h

`#include <QtCore> #include "tdiffusionmessages.h" #include "hautparleur.h"` Graphe des dépendances par inclusion de tinformationvoyageur.h :



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



Classes

– class TInformationVoyageur
Classe qui informe les voyageurs et qui gère la detection des arrêts.

Macros

– #define PI 3.1415
– #define PAS_EN_COURSE 0
Etat du bus : PAS_EN_COURSE pour l'attribut enCourse.
– #define EN_COURSE 1
Etat du bus : EN_COURSE pour l'attribut enCourse.
– #define COURSE_TERMINEE 2
Etat du bus : COURSE_TERMINEE pour l'attribut enCourse.

10.28.1 Documentation des macros

10.28.1.1 #define COURSE_TERMINEE 2

10.28.1.2 #define EN_COURSE 1

10.28.1.3 #define PAS_EN_COURSE 0

10.28.1.4 #define PI 3.1415

Référencé par TInformationVoyageur : :calculerDistance().

Index