ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

HONORIS UNITED UNIVERSITIES

# Rapport de Projet

3-ème Année

Ingénieurie Informatique et Réseaux

# Gestion d'une Flotte de Véhicules en C++

*Réalisé par: Lhoussaine GHALLOU*

*Encadré par: Mariame AMINE*

## Description du projet :

Le projet consiste en une application de gestion des véhicules électriques pour une entreprise, où l'on modélise divers objets et fonctionnalités autour de la gestion des employés, des véhicules, des flottes et des informations associées à chaque entité. Il s'agit d'une solution qui permet aux entreprises de gérer les réservations de véhicules électriques, l'affectation des véhicules à des employés, ainsi que le suivi du kilométrage et de la disponibilité des véhicules.

## Objectifs :

### 1. **Modélisation des entités**:

- Le projet implique la création de classes représentant des entités clés comme les employés, les véhicules, les flottes, et les cartes d'identité des employés. Chaque entité a des attributs et des méthodes pour gérer ses propriétés et comportements spécifiques.

### 2. **Gestion des employés**:

- Un des objectifs principaux est de permettre à l'entreprise de gérer ses employés et leur attribution à des véhicules. Les employés peuvent réserver des véhicules, et la gestion des réservations est effectuée par l'intermédiaire des classes Employee et HRManager.

### 3. **Gestion des véhicules**:

- Les véhicules sont modélisés avec des spécifications comme la marque, le modèle, le kilométrage, et l'état de réservation. Les véhicules peuvent être électriques et posséder des attributs spécifiques liés à l'autonomie et à la consommation.

### 4. **Gestion des flottes de véhicules**:

- Le projet permet de gérer une flotte de véhicules électriques, assigner des véhicules à des employés et suivre l'état de chaque véhicule. La classe Fleet et son gestionnaire FleetManager sont responsables de cette gestion.

### 5. **Réservation et affectation des véhicules**:

- Les employés peuvent réserver des véhicules, mais la réservation doit respecter certaines règles (par exemple, un véhicule ne peut être réservé qu'une seule fois). Le système doit

permettre à l'administrateur (HRManager) de gérer ces réservations et de mettre à jour les informations relatives aux véhicules réservés.

### 6. **Suivi du kilométrage et des réservations**:

- Un autre objectif important est de suivre le kilométrage des véhicules et de permettre à l'entreprise de mettre à jour ces informations régulièrement. Les données concernant le kilométrage et l'état des réservations sont cruciales pour la gestion de la flotte.

## Fonctionnalités clés :

- **Création et gestion des cartes d'identité des employés** (via la classe EmployeeIDCard).
- **Gestion des informations des véhicules** (via Vehicule et ElectricVehicule).
- **Suivi des réservations de véhicules**.
- **Gestion de la flotte de véhicules** (avec Fleet et FleetManager).
- **Gestion des employés** : embauche et licenciement, avec vérification des identifiants uniques et gestion des affectations.

## Objectifs à long terme :

L'objectif à long terme du projet est de créer un système robuste permettant aux entreprises de gérer efficacement leur flotte de véhicules électriques, avec une interface utilisateur intuitive pour les employés et une gestion complète des réservations et des affectations pour les responsables des ressources humaines.

# Chapter 1

# Hierarchical Index

## 1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# Chapter 2

# Class Index

## 2.1 Class List

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

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all files with brief descriptions:

# Chapter 4

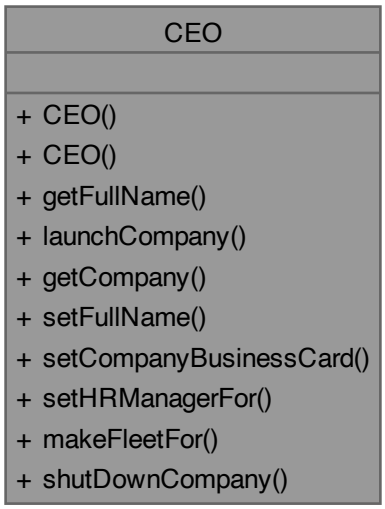
# Class Documentation

## 4.1 CEO Class Reference

Représente un PDG (Chief Executive Officer) qui gère plusieurs entreprises.

`#include <CEO.h>`

Collaboration diagram for CEO:




### Public Member Functions

- CEO ()=default

  *Constructeur par défaut de la classe CEO.*
- CEO (std::string fullName)

*Constructeur paramétré de la classe CEO.*

- const std::string & getFullName () const

  *Obtient le nom complet du PDG.*
- void launchCompany (CompanyBusinessCard IDCard)

  *Lance une nouvelle entreprise en utilisant une carte professionnelle.*
- std::shared_ptr< Company > getCompany (const std::string &name) const

  *Récupère une entreprise par son nom.*
- void setFullName (std::string fullName)

  *Définit le nom complet du PDG.*
- void setCompanyBusinessCard (const std::string &companyName, CompanyBusinessCard businessCard)

  *Met à jour la carte professionnelle d'une entreprise.*
- void setHRManagerFor (HRManager &&managerOfHR, const std::string &name)

  *Associe un gestionnaire des ressources humaines (HRManager) à une entreprise.*
- void makeFleetFor (Fleet &&fleet, const std::string &name)

  *Crée une flotte de véhicules pour une entreprise.*
- void shutDownCompany (const std::string &name)

  *Ferme une entreprise existante.*

**Friends**

- std::istream & operator>> (std::istream &is, CEO &ceo)

  *Surcharge de l'opérateur >> pour saisir les informations d'un PDG.*

### 4.1.1 Detailed Description

Représente un PDG (Chief Executive Officer) qui gère plusieurs entreprises.

La classe CEO permet de gérer des entreprises, leurs cartes professionnelles, ainsi que les gestionnaires de ressources humaines (HRManager) associés à chaque entreprise.

### 4.1.2 Constructor & Destructor Documentation

#### 4.1.2.1 CEO() [1/2]

```
CEO::CEO ()  [default]
```

Constructeur par défaut de la classe CEO.

#### 4.1.2.2 CEO() [2/2]

```
CEO::CEO (
            std::string fullName)  [inline], [explicit]
```

Constructeur paramétré de la classe CEO.

**Parameters**

| | |
|---|---|
| *fullName* | Le nom complet du PDG. |

### 4.1.3 Member Function Documentation

#### 4.1.3.1 getCompany()

```
std::shared_ptr< Company > CEO::getCompany (
            const std::string & name) const  [nodiscard]
```

Récupère une entreprise par son nom.

Recherche et retourne une entreprise par son nom.

**Parameters**

| | |
|---|---|
| *name* | Le nom de l'entreprise à rechercher. |

**Returns**

Un pointeur partagé vers l'entreprise si elle est trouvée, nullptr sinon.

**Parameters**

| | |
|---|---|
| *name* | Le nom de l'entreprise à rechercher. |

**Returns**

Un pointeur partagé vers l'entreprise trouvée.

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | Si l'entreprise n'est pas trouvée. |

Un `std::shared_ptr<Company>` est retourné pour éviter la gestion manuelle de la mémoire tout en permettant le partage de la propriété de l'objet. La copie de l'objet est évitée, ce qui permet d'améliorer l'efficacité.

#### 4.1.3.2 getFullName()

```
const std::string & CEO::getFullName () const  [nodiscard]
```

Obtient le nom complet du PDG.

**Returns**

Une référence constante à une chaîne représentant le nom complet.

Le nom complet du PDG sous forme de chaîne de caractères.

Utilisation d'un `const` pour éviter les copies inutiles, car le nom complet ne doit pas être modifié ici.

#### 4.1.3.3 launchCompany()

```
void CEO::launchCompany (
            CompanyBusinessCard IDCard)
```

Lance une nouvelle entreprise en utilisant une carte professionnelle.

Lance une nouvelle entreprise sous la gestion du PDG.

**Parameters**

| | |
|---|---|
| *IDCard* | La carte professionnelle associée à la nouvelle entreprise. |
| *IDCard* | La carte professionnelle de l'entreprise à créer. |

`IDCard` est passé par valeur et déplacé dans la nouvelle entreprise. Cela évite une copie inutile de `IDCard` lors de l'appel de `std::make_shared`.

#### 4.1.3.4 makeFleetFor()

```
void CEO::makeFleetFor (
            Fleet && fleet,
            const std::string & name)
```

Crée une flotte de véhicules pour une entreprise.

**Parameters**

| | |
|---|---|
| *fleet* | L'objet représentant la flotte de véhicules. |
| *name* | Le nom de l'entreprise pour laquelle la flotte est créée. |
| *fleet* | La flotte de véhicules à associer à l'entreprise. |

`fleet` est passé par rvalue et déplacé pour éviter une copie coûteuse de la flotte.

#### 4.1.3.5 setCompanyBusinessCard()

```
void CEO::setCompanyBusinessCard (
            const std::string & companyName,
            CompanyBusinessCard businessCard)
```

Met à jour la carte professionnelle d'une entreprise.

Définit la carte professionnelle d'une entreprise gérée par le PDG.

**Parameters**

| | |
|---|---|
| *companyName* | Le nom de l'entreprise concernée. |
| *businessCard* | La nouvelle carte professionnelle associée à l'entreprise. |
| *companyName* | Le nom de l'entreprise dont la carte professionnelle est mise à jour. |
| *businessCard* | La nouvelle carte professionnelle de l'entreprise. |

L'argument `businessCard` est passé par valeur et déplacé à l'intérieur de la méthode. Cela permet de transférer la possession de l'objet sans effectuer de copie.

#### 4.1.3.6 setFullName()

```
void CEO::setFullName (
            std::string fullName)
```

Définit le nom complet du PDG.

**Parameters**

| | |
|---|---|
| *fullName* | Le nouveau nom complet du PDG. |
| *fullName* | Le nom complet à attribuer au PDG. |

L'argument `fullName` est passé par valeur, mais il est déplacé (`std::move`) pour éviter la copie inutile si l'argument est une rvalue. Cela améliore la performance en réduisant les copies.

#### 4.1.3.7 setHRManagerFor()

```
void CEO::setHRManagerFor (
            HRManager && managerOfHR,
            const std::string & name)
```

Associe un gestionnaire des ressources humaines (HRManager) à une entreprise.

**Parameters**

| | |
|---|---|
| *managerOfHR* | Le gestionnaire des ressources humaines. |
| *name* | Le nom de l'entreprise concernée. |
| *managerOfHR* | Le gestionnaire des ressources humaines à affecter. |
| *name* | Le nom de l'entreprise pour laquelle le gestionnaire RH est affecté. |

L'argument `managerOfHR` est un rvalue, il est donc déplacé dans la méthode. Le déplacement permet de transférer l'objet sans effectuer une copie coûteuse.

#### 4.1.3.8 shutDownCompany()

```
void CEO::shutDownCompany (
            const std::string & name)
```

Ferme une entreprise existante.

Ferme une entreprise gérée par le PDG.

**Parameters**

| | |
|---|---|
| *name* | Le nom de l'entreprise à fermer. |
| *name* | Le nom de l'entreprise à fermer. |

Le nom de l'entreprise est utilisé pour retrouver l'entreprise dans la liste, sans nécessiter de copies supplémentaires. La méthode `remove_if` permet de supprimer l'entreprise sans créer de nouvelles copies des objets dans la liste.

### 4.1.4 Friends And Related Symbol Documentation

#### 4.1.4.1 operator>>

```
std::istream & operator>> (
            std::istream & is,
            CEO & ceo)  [friend]
```

Surcharge de l'opérateur >> pour saisir les informations d'un PDG.

**Parameters**

| | |
|---|---|
| *is* | Le flux d'entrée. |
| *ceo* | L'objet CEO à remplir. |

**Returns**

Une référence vers le flux d'entrée.

**Parameters**

| | |
|---|---|
| *is* | Le flux d'entrée pour saisir les informations. |
| *ceo* | L'objet PDG à remplir. |

**Returns**

Une référence vers le flux d'entrée.

Le flux d'entrée est utilisé pour remplir l'objet `ceo`. Aucune copie n'est nécessaire, car on travaille directement avec la référence de l'objet `ceo`.

The documentation for this class was generated from the following files:

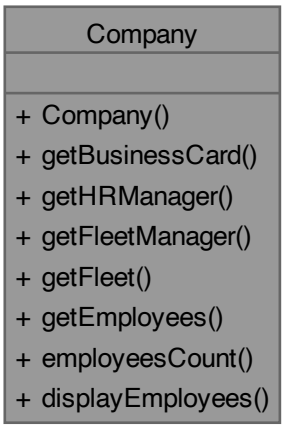
- include/CEO.h
- src/CEO.cpp

## 4.2 Company Class Reference

Représente une entreprise avec ses employés, sa flotte de véhicules, et son gestionnaire RH.

`#include <Company.h>`

Collaboration diagram for Company:



### Public Member Functions

- Company (CompanyBusinessCard businessCard)

  *Constructeur paramétré de la classe Company.*
- const CompanyBusinessCard & getBusinessCard () const

  *Obtient la carte professionnelle de l'entreprise.*
- std::shared_ptr< HRManager > getHRManager () const

  *Obtient le gestionnaire des ressources humaines (HRManager) de l'entreprise.*
- std::shared_ptr< FleetManager > getFleetManager () const

  *Obtient le gestionnaire de la flotte de véhicules.*
- std::shared_ptr< Fleet > getFleet () const

  *Obtient la flotte de véhicules de l'entreprise.*
- std::shared_ptr< std::vector< std::shared_ptr< Employee > > > getEmployees () const

  *Obtient la liste des employés de l'entreprise.*
- size_t employeesCount () const

  *Retourne le nombre d'employés dans l'entreprise.*
- void displayEmployees () const

  *Affiche les informations de tous les employés de l'entreprise.*

### Friends

- class CEO
- class HRManager

## 4.2.1 Detailed Description

Représente une entreprise avec ses employés, sa flotte de véhicules, et son gestionnaire RH.

Cette classe gère les informations essentielles d'une entreprise, y compris la carte professionnelle, les employés, le gestionnaire des ressources humaines (HRManager), le gestionnaire de flotte (FleetManager), et la flotte de véhicules.

## 4.2.2 Constructor & Destructor Documentation

### 4.2.2.1 Company()

```
Company::Company (
            CompanyBusinessCard businessCard)  [inline], [explicit]
```

Constructeur paramétré de la classe Company.

**Parameters**

| | |
|---|---|
| *businessCard* | La carte professionnelle de l'entreprise. |

### 4.2.3 Member Function Documentation

#### 4.2.3.1 displayEmployees()

```
void Company::displayEmployees () const
```

Affiche les informations de tous les employés de l'entreprise.

Affiche les informations des employés de l'entreprise.

La première méthode d'affichage affiche la carte d'identité du premier employé, et les autres employés sont affichés dans une boucle. Aucune copie d'objet n'est effectuée, nous accédons directement aux éléments du vecteur et utilisons `getIDCARD()` pour afficher les informations des employés.

#### 4.2.3.2 employeesCount()

```
size_t Company::employeesCount () const  [nodiscard]
```

Retourne le nombre d'employés dans l'entreprise.

**Returns**

Le nombre d'employés sous forme d'un entier de type size_t.

Le nombre d'employés sous forme d'un entier de type `size_t`.

Utiliser la méthode `size()` sur le vecteur des employés permet d'obtenir rapidement le nombre d'employés sans effectuer de copie du vecteur lui-même.

#### 4.2.3.3 getBusinessCard()

```
const CompanyBusinessCard & Company::getBusinessCard () const  [nodiscard]
```

Obtient la carte professionnelle de l'entreprise.

**Returns**

Une référence constante vers l'objet CompanyBusinessCard.

Une référence constante vers l'objet CompanyBusinessCard.

La carte professionnelle est retournée par référence constante pour éviter une copie inutile de l'objet, étant donné que l'objet est immuable dans cette méthode.

#### 4.2.3.4 getEmployees()

```
std::shared_ptr< std::vector< std::shared_ptr< Employee > > > Company::getEmployees () const
[nodiscard]
```

Obtient la liste des employés de l'entreprise.

**Returns**

Un pointeur partagé vers un vecteur contenant les employés.

Un pointeur partagé vers le vecteur contenant les employés.

Utiliser un `std::shared_ptr<std::vector<std::shared_ptr<Employee>>>` permet de partager le vecteur des employés sans copier les objets ou le vecteur lui-même.

#### 4.2.3.5 getFleet()

```
std::shared_ptr< Fleet > Company::getFleet () const  [nodiscard]
```

Obtient la flotte de véhicules de l'entreprise.

**Returns**

Un pointeur partagé vers Fleet.

Un pointeur partagé vers la flotte de véhicules.

Le retour d'un `std::shared_ptr<Fleet>` permet de partager la gestion de la flotte sans effectuer de copie de l'objet.

#### 4.2.3.6 getFleetManager()

```
std::shared_ptr< FleetManager > Company::getFleetManager () const  [nodiscard]
```

Obtient le gestionnaire de la flotte de véhicules.

Obtient le gestionnaire de flotte de l'entreprise.

**Returns**

Un pointeur partagé vers FleetManager.

Un pointeur partagé vers le gestionnaire de flotte.

Comme pour `getHRManager`, l'utilisation de `std::shared_ptr` permet de partager l'objet sans créer de copie, ce qui est plus efficace.

#### 4.2.3.7 getHRManager()

```
std::shared_ptr< HRManager > Company::getHRManager () const  [nodiscard]
```

Obtient le gestionnaire des ressources humaines (HRManager) de l'entreprise.

Obtient le gestionnaire des ressources humaines de l'entreprise.

**Returns**

Un pointeur partagé vers HRManager.

Un pointeur partagé vers le gestionnaire des ressources humaines.

Le gestionnaire des ressources humaines est retourné sous forme de `std::shared_ptr` pour permettre une gestion partagée de la mémoire sans effectuer de copie de l'objet.

### 4.2.4 Friends And Related Symbol Documentation

#### 4.2.4.1 CEO

```
friend class CEO  [friend]
```

Permet à la classe CEO d'accéder aux membres privés.

#### 4.2.4.2 HRManager

```
friend class HRManager  [friend]
```

Permet à la classe HRManager d'accéder aux membres privés.

The documentation for this class was generated from the following files:

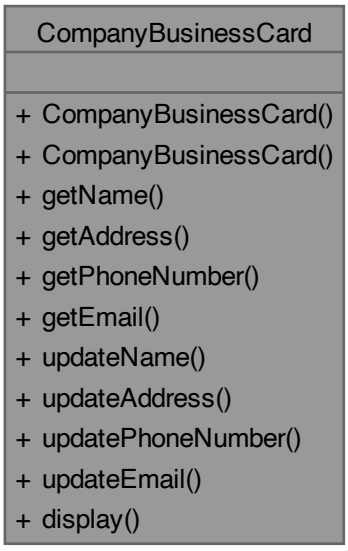
- include/Company.h
- src/Company.cpp

## 4.3 CompanyBusinessCard Struct Reference

Représente une carte professionnelle pour une entreprise.

```
#include <CompanyBusinessCard.h>
```

Collaboration diagram for CompanyBusinessCard:



**Public Member Functions**

- CompanyBusinessCard ()=default
  *Constructeur par défaut.*
- CompanyBusinessCard (std::string name, std::string address, std::string phoneNumber, std::string email)
  *Constructeur paramétré pour initialiser toutes les informations de la carte.*
- const std::string & getName () const
  *Obtient le nom de l'entreprise.*
- const std::string & getAddress () const
  *Obtient l'adresse de l'entreprise.*
- const std::string & getPhoneNumber () const
  *Obtient le numéro de téléphone de l'entreprise.*
- const std::string & getEmail () const
  *Obtient l'adresse e-mail de l'entreprise.*
- void updateName (std::string newName)
  *Met à jour le nom de l'entreprise.*
- void updateAddress (std::string newAddress)
  *Met à jour l'adresse de l'entreprise.*
- void updatePhoneNumber (std::string newPhoneNumber)
  *Met à jour le numéro de téléphone de l'entreprise.*
- void updateEmail (std::string newEmail)
  *Met à jour l'adresse e-mail de l'entreprise.*
- void display () const
  *Affiche les informations de la carte professionnelle.*

**Friends**

- std::ostream & operator<< (std::ostream &os, const CompanyBusinessCard &BC)
  *Surcharge de l'opérateur << pour afficher la carte professionnelle.*
- std::istream & operator>> (std::istream &is, CompanyBusinessCard &BC)
  *Surcharge de l'opérateur >> pour saisir les informations de la carte.*

### 4.3.1 Detailed Description

Représente une carte professionnelle pour une entreprise.

Cette structure contient les informations de base liées à une entreprise, telles que son nom, son adresse, son numéro de téléphone et son adresse e-mail.

### 4.3.2 Constructor & Destructor Documentation

#### 4.3.2.1 CompanyBusinessCard() [1/2]

```
CompanyBusinessCard::CompanyBusinessCard ()  [default]
```

Constructeur par défaut.

#### 4.3.2.2 CompanyBusinessCard() [2/2]

```
CompanyBusinessCard::CompanyBusinessCard (
            std::string name,
            std::string address,
            std::string phoneNumber,
            std::string email)  [inline]
```

Constructeur paramétré pour initialiser toutes les informations de la carte.

**Parameters**

| | |
|---|---|
| *name* | Le nom de l'entreprise. |
| *address* | L'adresse de l'entreprise. |
| *phoneNumber* | Le numéro de téléphone de l'entreprise. |
| *email* | L'adresse e-mail de l'entreprise. |

### 4.3.3 Member Function Documentation

#### 4.3.3.1 display()

```
void CompanyBusinessCard::display () const
```

Affiche les informations de la carte professionnelle.

Affiche les informations de la carte professionnelle de l'entreprise.

La méthode `display()` affiche un en-tête formaté avec des colonnes pour chaque champ de la carte professionnelle et utilise l'opérateur << pour afficher les données formatées.

#### 4.3.3.2 getAddress()

```
const std::string & CompanyBusinessCard::getAddress () const  [nodiscard]
```

Obtient l'adresse de l'entreprise.

**Returns**

Une référence constante vers l'adresse.

Une référence constante vers l'adresse de l'entreprise.

Comme pour `getName()`, une référence constante est retournée pour éviter la copie inutile de l'objet `address`.

#### 4.3.3.3 getEmail()

```
const std::string & CompanyBusinessCard::getEmail () const  [nodiscard]
```

Obtient l'adresse e-mail de l'entreprise.

**Returns**

Une référence constante vers l'adresse e-mail.

Une référence constante vers l'adresse e-mail de l'entreprise.

Retourner une référence constante pour optimiser l'accès sans créer de copie supplémentaire des données.

#### 4.3.3.4 getName()

```
const std::string & CompanyBusinessCard::getName () const  [nodiscard]
```

Obtient le nom de l'entreprise.

**Returns**

Une référence constante vers le nom.

Une référence constante vers le nom de l'entreprise.

La méthode retourne une référence constante pour éviter une copie inutile de l'objet `name`, car il n'y a aucune modification effectuée sur la donnée.

#### 4.3.3.5 getPhoneNumber()

```
const std::string & CompanyBusinessCard::getPhoneNumber () const  [nodiscard]
```

Obtient le numéro de téléphone de l'entreprise.

**Returns**

Une référence constante vers le numéro de téléphone.

Une référence constante vers le numéro de téléphone de l'entreprise.

Cette méthode utilise une référence constante pour le même principe : éviter une copie inutile des informations de contact.

#### 4.3.3.6 updateAddress()

```
void CompanyBusinessCard::updateAddress (
            std::string newAddress)
```

Met à jour l'adresse de l'entreprise.

**Parameters**

| | |
|---|---|
| *newAddress* | La nouvelle adresse de l'entreprise. |
| *newAddress* | La nouvelle adresse de l'entreprise. |

`newAddress` est déplacé pour optimiser l'affectation, en évitant une copie.

#### 4.3.3.7 updateEmail()

```
void CompanyBusinessCard::updateEmail (
            std::string newEmail)
```

Met à jour l'adresse e-mail de l'entreprise.

**Parameters**

| | |
|---|---|
| *newEmail* | La nouvelle adresse e-mail. |
| *newEmail* | La nouvelle adresse e-mail de l'entreprise. |

L'argument `newEmail` est déplacé afin de transférer directement l'objet sans créer une copie inutile.

#### 4.3.3.8 updateName()

```
void CompanyBusinessCard::updateName (
            std::string newName)
```

Met à jour le nom de l'entreprise.

**Parameters**

| | |
|---|---|
| *newName* | Le nouveau nom de l'entreprise. |
| *newName* | Le nouveau nom de l'entreprise. |

L'argument `newName` est passé par valeur et est déplacé dans l'objet `name`. Utiliser `std::move` permet de transférer la propriété de l'objet sans créer de copie.

#### 4.3.3.9 updatePhoneNumber()

```
void CompanyBusinessCard::updatePhoneNumber (
            std::string newPhoneNumber)
```

Met à jour le numéro de téléphone de l'entreprise.

**Parameters**

| | |
|---|---|
| *newPhoneNumber* | Le nouveau numéro de téléphone. |
| *newPhoneNumber* | Le nouveau numéro de téléphone de l'entreprise. |

Comme pour `updateName` et `updateAddress`, l'argument est déplacé pour éviter une copie.

### 4.3.4 Friends And Related Symbol Documentation

#### 4.3.4.1 operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const CompanyBusinessCard & BC)  [friend]
```

Surcharge de l'opérateur << pour afficher la carte professionnelle.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie. |
| *BC* | La carte professionnelle à afficher. |

**Returns**

Une référence vers le flux de sortie.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les informations seront imprimées. |
| *BC* | La carte professionnelle à afficher. |

**Returns**

Une référence vers le flux de sortie.

L'opérateur << est utilisé pour afficher une ligne formatée de la carte professionnelle. Aucune copie d'objet n'est effectuée ici, juste un accès direct aux membres de `CompanyBusinessCard`.

#### 4.3.4.2 operator>>

```
std::istream & operator>> (
            std::istream & is,
            CompanyBusinessCard & BC)  [friend]
```

Surcharge de l'opérateur >> pour saisir les informations de la carte.

**Parameters**

| | |
|---|---|
| *is* | Le flux d'entrée. |
| *BC* | La carte professionnelle à remplir. |

**Returns**

Une référence vers le flux d'entrée.

**Parameters**

| | |
|---|---|
| *is* | Le flux d'entrée où les informations seront lues. |
| *BC* | La carte professionnelle à remplir. |

**Returns**

Une référence vers le flux d'entrée.

Cet opérateur permet de saisir les informations de la carte professionnelle en utilisant un flux d'entrée. Chaque champ est lu et stocké dans les attributs respectifs de `CompanyBusinessCard`.

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

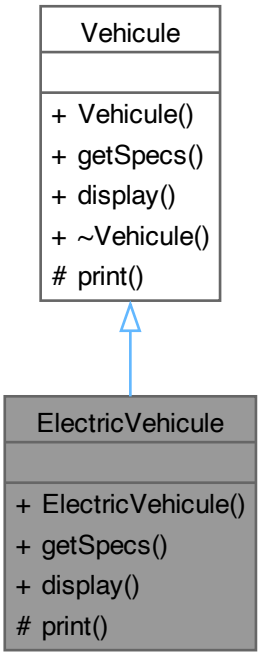
- include/CompanyBusinessCard.h
- src/CompanyBusinessCard.cpp

## 4.4 ElectricVehicule Class Reference

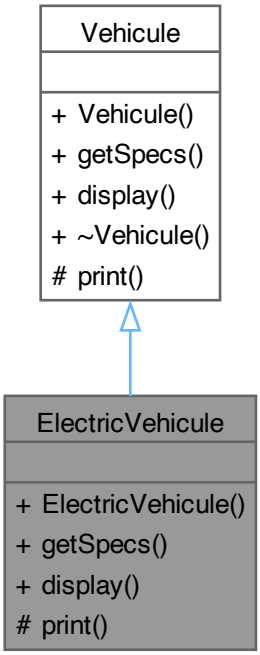
Représente un véhicule électrique, dérivé de la classe Vehicule.

`#include <ElectricVehicule.h>`

Inheritance diagram for ElectricVehicule:



Collaboration diagram for ElectricVehicule:




### Public Member Functions

- ElectricVehicule (ElectricVehiculeSpecs specs)

  *Constructeur paramétré pour initialiser un véhicule électrique avec ses spécifications.*
- const ElectricVehiculeSpecs & getSpecs () override

  *Obtient les spécifications du véhicule électrique.*
- void display () const override

  *Affiche les détails du véhicule électrique.*

## Public Member Functions inherited from Vehicule

- Vehicule (VehiculeSpecs specs)

  *Constructeur paramétré pour initialiser un véhicule avec ses spécifications.*
- virtual ~Vehicule ()=default

  *Destructeur virtuel pour garantir la destruction correcte des objets dérivés.*

### Protected Member Functions

- void print (std::ostream &os) const override

  *Méthode protégée pour imprimer les informations du véhicule électrique dans un flux de sortie.*

## Protected Member Functions inherited from Vehicule

### Friends

- std::ostream & operator<< (std::ostream &os, const ElectricVehicule &eVehicule)

  *Surcharge de l'opérateur << pour afficher un véhicule électrique.*

### 4.4.1 Detailed Description

Représente un véhicule électrique, dérivé de la classe Vehicule.

Cette classe étend les fonctionnalités de la classe Vehicule en ajoutant des spécifications spécifiques aux véhicules électriques, encapsulées dans la classe ElectricVehiculeSpecs.

### 4.4.2 Constructor & Destructor Documentation

#### 4.4.2.1 ElectricVehicule()

```
ElectricVehicule::ElectricVehicule (
            ElectricVehiculeSpecs specs)  [inline], [explicit]
```

Constructeur paramétré pour initialiser un véhicule électrique avec ses spécifications.

**Parameters**

| | |
|---|---|
| *specs* | Les spécifications du véhicule électrique encapsulées dans ElectricVehiculeSpecs. |

### 4.4.3 Member Function Documentation

#### 4.4.3.1 display()

```
void ElectricVehicule::display () const  [override], [virtual]
```

Affiche les détails du véhicule électrique.

Affiche les spécifications du véhicule électrique.

Cette méthode redéfinit la méthode virtuelle de la classe de base pour afficher les informations spécifiques au véhicule électrique.

Appelle la méthode `display()` des spécifications pour afficher les informations du véhicule.

Reimplemented from Vehicule.

#### 4.4.3.2 getSpecs()

```
const ElectricVehiculeSpecs & ElectricVehicule::getSpecs ()  [nodiscard], [override], [virtual]
```

Obtient les spécifications du véhicule électrique.

**Returns**

Une référence constante vers l'objet ElectricVehiculeSpecs.

Une référence constante vers les spécifications du véhicule électrique.

Retourne les spécifications par référence pour une performance optimale.

Reimplemented from Vehicule.

#### 4.4.3.3 print()

```
void ElectricVehicule::print (
            std::ostream & os) const  [override], [protected], [virtual]
```

Méthode protégée pour imprimer les informations du véhicule électrique dans un flux de sortie.

Imprime les spécifications du véhicule électrique dans un flux de sortie.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les informations sont imprimées. |
| *os* | Le flux de sortie où les spécifications du véhicule seront imprimées. |

Cette méthode utilise l'opérateur << pour afficher directement les spécifications du véhicule électrique.

Reimplemented from Vehicule.

### 4.4.4 Friends And Related Symbol Documentation

#### 4.4.4.1 operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const ElectricVehicule & eVehicule)  [friend]
```

Surcharge de l'opérateur << pour afficher un véhicule électrique.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les informations du véhicule sont imprimées. |
| *eVehicule* | Le véhicule électrique à afficher. |

**Returns**

Une référence vers le flux de sortie.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les informations du véhicule seront imprimées. |
| *eVehicule* | Le véhicule électrique à afficher. |

**Returns**

Une référence vers le flux de sortie.

Utilise `print()` pour afficher les détails du véhicule dans le flux.

The documentation for this class was generated from the following files:

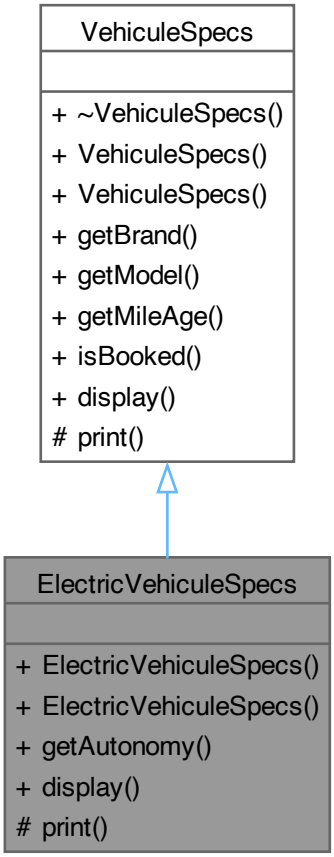
- include/ElectricVehicule.h
- src/ElectricVehicule.cpp

## 4.5 ElectricVehiculeSpecs Struct Reference

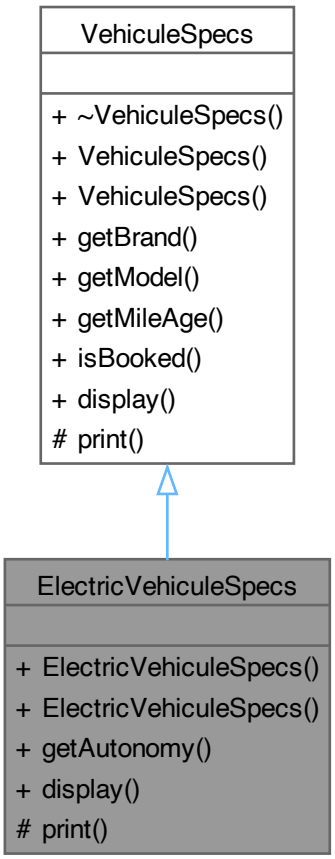
Représente les spécifications spécifiques d'un véhicule électrique.

`#include <ElectricVehiculeSpecs.h>`

Inheritance diagram for ElectricVehiculeSpecs:



Collaboration diagram for ElectricVehiculeSpecs:




### Public Member Functions

- ElectricVehiculeSpecs (std::string brand, std::string model, int autonomy)

  *Constructeur paramétré pour initialiser les spécifications d'un véhicule électrique.*
- ElectricVehiculeSpecs (std::string brand, std::string model, unsigned int mileAge, bool booked, int autonomy)

  *Constructeur paramétré pour initialiser toutes les spécifications, y compris le kilométrage et l'état de réservation.*
- int getAutonomy () const

  *Obtient l'autonomie du véhicule électrique.*
- void display () const override

  *Affiche les spécifications du véhicule électrique.*

## Public Member Functions inherited from VehiculeSpecs

- virtual ~VehiculeSpecs ()=default

  *Destructeur virtuel pour permettre la destruction correcte des objets dérivés.*
- VehiculeSpecs (std::string brand, std::string model)

  *Constructeur pour initialiser la marque et le modèle du véhicule.*

- VehiculeSpecs (std::string brand, std::string model, unsigned int mileAge, bool booked)

  *Constructeur pour initialiser toutes les spécifications du véhicule.*
- const std::string & getBrand () const

  *Obtient la marque du véhicule.*
- const std::string & getModel () const

  *Obtient le modèle du véhicule.*
- unsigned int getMileAge () const

  *Obtient le kilométrage du véhicule.*
- bool isBooked () const

  *Vérifie si le véhicule est réservé.*

#### Protected Member Functions

- void print (std::ostream &os) const override

  *Méthode protégée pour imprimer les informations des spécifications dans un flux de sortie.*

### Protected Member Functions inherited from VehiculeSpecs

#### Friends

- class FleetManager
- std::ostream & operator<< (std::ostream &os, const ElectricVehiculeSpecs &specs)

  *Surcharge de l'opérateur << pour afficher les spécifications d'un véhicule électrique.*

## 4.5.1 Detailed Description

Représente les spécifications spécifiques d'un véhicule électrique.

Cette structure hérite de VehiculeSpecs et ajoute un attribut spécifique pour l'autonomie des véhicules électriques. Elle inclut des méthodes pour afficher et manipuler ces spécifications.

## 4.5.2 Constructor & Destructor Documentation

### 4.5.2.1 ElectricVehiculeSpecs() [1/2]

```
ElectricVehiculeSpecs::ElectricVehiculeSpecs (
            std::string brand,
            std::string model,
            int autonomy)  [inline]
```

Constructeur paramétré pour initialiser les spécifications d'un véhicule électrique.

**Parameters**

| | |
|---|---|
| *brand* | La marque du véhicule. |
| *model* | Le modèle du véhicule. |
| *autonomy* | L'autonomie du véhicule électrique en kilomètres. |

#### 4.5.2.2 ElectricVehiculeSpecs() [2/2]

```
ElectricVehiculeSpecs::ElectricVehiculeSpecs (
            std::string brand,
            std::string model,
            unsigned int mileAge,
            bool booked,
            int autonomy)  [inline]
```

Constructeur paramétré pour initialiser toutes les spécifications, y compris le kilométrage et l'état de réservation.

**Parameters**

| | |
|---|---|
| *brand* | La marque du véhicule. |
| *model* | Le modèle du véhicule. |
| *mileAge* | Le kilométrage du véhicule. |
| *booked* | L'état de réservation du véhicule (true si réservé, false sinon). |
| *autonomy* | L'autonomie du véhicule électrique en kilomètres. |

### 4.5.3 Member Function Documentation

#### 4.5.3.1 display()

```
void ElectricVehiculeSpecs::display () const  [override], [virtual]
```

Affiche les spécifications du véhicule électrique.

Affiche les spécifications du véhicule électrique de manière lisible.

Cette méthode redéfinit la méthode virtuelle de la classe de base pour inclure les informations spécifiques à l'autonomie.

Affiche un en-tête suivi des spécifications du véhicule dans un format lisible, en utilisant l'opérateur $<<$ pour afficher les informations.

Reimplemented from VehiculeSpecs.

#### 4.5.3.2 getAutonomy()

```
int ElectricVehiculeSpecs::getAutonomy () const  [nodiscard]
```

Obtient l'autonomie du véhicule électrique.

**Returns**

L'autonomie en kilomètres sous forme d'un entier.

L'autonomie en kilomètres.

Retourne la valeur de l'autonomie, sans modifier l'objet, pour un accès efficace.

#### 4.5.3.3 print()

```
void ElectricVehiculeSpecs::print (
            std::ostream & os) const  [override], [protected], [virtual]
```

Méthode protégée pour imprimer les informations des spécifications dans un flux de sortie.

Imprime les spécifications du véhicule électrique dans un flux de sortie.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les informations sont imprimées. |
| *os* | Le flux de sortie où les spécifications seront imprimées. |

Affiche les spécifications du véhicule, y compris la marque, le modèle, le kilométrage, l'état de réservation et l'autonomie, dans un format bien structuré.

Reimplemented from VehiculeSpecs.

### 4.5.4 Friends And Related Symbol Documentation

#### 4.5.4.1 FleetManager

```
friend class FleetManager  [friend]
```

Permet à la classe FleetManager d'accéder aux membres privés.

#### 4.5.4.2 operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const ElectricVehiculeSpecs & specs)  [friend]
```

Surcharge de l'opérateur << pour afficher les spécifications d'un véhicule électrique.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les informations sont imprimées. |
| *specs* | Les spécifications du véhicule électrique à afficher. |

**Returns**

Une référence vers le flux de sortie.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les spécifications seront imprimées. |
| *specs* | Les spécifications du véhicule à afficher. |

**Returns**

Une référence vers le flux de sortie.

L'opérateur << est utilisé pour afficher les spécifications du véhicule en appelant la méthode `print()`. Cela permet de conserver un affichage structuré.

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

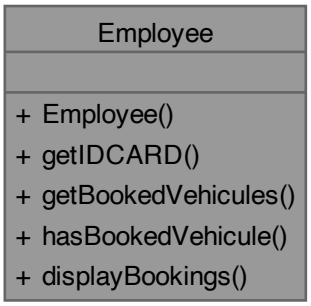
- include/ElectricVehiculeSpecs.h
- src/ElectricVehiculeSpecs.cpp

## 4.6 Employee Class Reference

Représente un employé dans le système de gestion de flotte.

`#include <Employee.h>`

Collaboration diagram for Employee:




### Public Member Functions

- Employee (EmployeeIDCard IDCard)
  *Constructeur paramétré pour initialiser un employé avec une carte d'identité.*
- EmployeeIDCard & getIDCARD ()
  *Obtient la carte d'identité de l'employé.*
- const std::vector< std::shared_ptr< Vehicule > > & getBookedVehicules () const
  *Obtient la liste des véhicules réservés par l'employé.*
- bool hasBookedVehicule (const std::shared_ptr< Vehicule > &vehicule) const
  *Vérifie si l'employé a réservé un véhicule donné.*
- void displayBookings () const
  *Affiche la liste des véhicules réservés par l'employé.*

### Friends

- class FleetManager

### 4.6.1 Detailed Description

Représente un employé dans le système de gestion de flotte.

Cette classe gère les informations d'identité d'un employé, représentées par une carte ID (EmployeeIDCard), ainsi que les véhicules qu'il a réservés.

### 4.6.2 Constructor & Destructor Documentation

#### 4.6.2.1 Employee()

```
Employee::Employee (
            EmployeeIDCard IDCard)  [inline], [explicit]
```

Constructeur paramétré pour initialiser un employé avec une carte d'identité.

**Parameters**

| | |
|---|---|
| *IDCard* | La carte d'identité de l'employé. |

### 4.6.3 Member Function Documentation

#### 4.6.3.1 displayBookings()

```
void Employee::displayBookings () const
```

Affiche la liste des véhicules réservés par l'employé.

Affiche les réservations de l'employé.

La méthode affiche d'abord le premier véhicule réservé, puis parcourt le reste de la liste pour afficher les autres véhicules réservés. Aucune copie d'objet n'est effectuée ici, les éléments sont accédés directement à partir du vecteur.

#### 4.6.3.2 getBookedVehicules()

```
const std::vector< std::shared_ptr< Vehicule > > & Employee::getBookedVehicules () const
[nodiscard]
```

Obtient la liste des véhicules réservés par l'employé.

Obtient les véhicules réservés par l'employé.

**Returns**

Une référence constante vers un vecteur contenant les véhicules réservés.

Une référence constante vers la liste des véhicules réservés.

Retourne la liste des véhicules réservés sous forme de référence constante pour éviter des copies inutiles tout en permettant une modification par d'autres parties du programme.

#### 4.6.3.3 getIDCARD()

```
EmployeeIDCard & Employee::getIDCARD ()  [nodiscard]
```

Obtient la carte d'identité de l'employé.

**Returns**

Une référence vers l'objet EmployeeIDCard.

Une référence vers la carte d'identité de l'employé.

Retourne la carte d'identité par référence pour éviter de faire une copie inutile de l'objet `EmployeeIDCard`. Cela permet une gestion efficace des ressources.

#### 4.6.3.4 hasBookedVehicule()

```
bool Employee::hasBookedVehicule (
            const std::shared_ptr< Vehicule > & vehicule) const
```

Vérifie si l'employé a réservé un véhicule donné.

Vérifie si l'employé a réservé un véhicule spécifique.

**Parameters**

| | |
|---|---|
| *vehicule* | Un pointeur partagé vers le véhicule à vérifier. |

**Returns**

`true` si le véhicule est réservé par l'employé, `false` sinon.

**Parameters**

| | |
|---|---|
| *vehicule* | Le véhicule à vérifier. |

**Returns**

`true` si le véhicule est réservé par l'employé, sinon `false`.

Cette méthode parcourt la liste des véhicules réservés par l'employé pour vérifier si le véhicule donné est présent. Le passage par référence évite une copie du véhicule.

### 4.6.4 Friends And Related Symbol Documentation

#### 4.6.4.1 FleetManager

```
friend class FleetManager  [friend]
```

Permet à la classe FleetManager d'accéder aux membres privés.

The documentation for this class was generated from the following files:

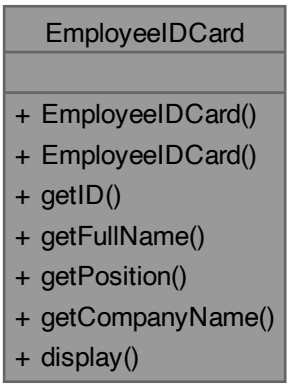
- include/Employee.h
- src/Employee.cpp

## 4.7 EmployeeIDCard Struct Reference

Représente une carte d'identité pour un employé.

`#include <EmployeeIDCard.h>`

Collaboration diagram for EmployeeIDCard:



**Public Member Functions**

- EmployeeIDCard ()=default

  *Constructeur par défaut.*
- EmployeeIDCard (std::string companyName, unsigned int ID, std::string fullName, std::string position)

  *Constructeur paramétré pour initialiser les informations de la carte d'identité.*
- double getID () const

  *Obtient le numéro d'identification de l'employé.*
- const std::string & getFullName () const

  *Obtient le nom complet de l'employé.*
- const std::string & getPosition () const

  *Obtient le poste de l'employé.*
- const std::string & getCompanyName () const

  *Obtient le nom de l'entreprise associée à l'employé.*
- void display () const

  *Affiche les informations de la carte d'identité.*

**Friends**

- class HRManager
- std::ostream & operator<< (std::ostream &os, const EmployeeIDCard &IDCard)

  *Surcharge de l'opérateur << pour afficher les informations de la carte d'identité.*
- std::istream & operator>> (std::istream &is, EmployeeIDCard &IDCard)

  *Surcharge de l'opérateur >> pour lire les informations de la carte d'identité.*

### 4.7.1 Detailed Description

Représente une carte d'identité pour un employé.

Cette structure contient les informations essentielles d'un employé, telles que le nom complet, le poste, le numéro d'identification (ID), et le nom de l'entreprise associée.

### 4.7.2 Constructor & Destructor Documentation

#### 4.7.2.1 EmployeeIDCard() [1/2]

```
EmployeeIDCard::EmployeeIDCard ()  [default]
```

Constructeur par défaut.

#### 4.7.2.2 EmployeeIDCard() [2/2]

```
EmployeeIDCard::EmployeeIDCard (
            std::string companyName,
            unsigned int ID,
            std::string fullName,
            std::string position)  [inline]
```

Constructeur paramétré pour initialiser les informations de la carte d'identité.

**Parameters**

| | |
|---|---|
| *companyName* | Le nom de l'entreprise. |
| *ID* | Le numéro d'identification de l'employé. |
| *fullName* | Le nom complet de l'employé. |
| *position* | Le poste de l'employé. |

### 4.7.3 Member Function Documentation

#### 4.7.3.1 display()

```
void EmployeeIDCard::display () const
```

Affiche les informations de la carte d'identité.

Affiche les informations de la carte d'identité de l'employé.

Affiche les informations de la carte d'identité dans un format lisible avec un en-tête. Cette méthode utilise l'opérateur $<<$ pour afficher les données formatées.

#### 4.7.3.2 getCompanyName()

```
const std::string & EmployeeIDCard::getCompanyName () const  [nodiscard]
```

Obtient le nom de l'entreprise associée à l'employé.

Obtient le nom de l'entreprise de l'employé.

**Returns**

Une référence constante vers le nom de l'entreprise.

Une référence constante vers le nom de l'entreprise.

Retourne une référence constante vers le nom de l'entreprise pour éviter une copie inutile.

#### 4.7.3.3 getFullName()

```
const std::string & EmployeeIDCard::getFullName () const  [nodiscard]
```

Obtient le nom complet de l'employé.

**Returns**

Une référence constante vers le nom complet.

Une référence constante vers le nom complet de l'employé.

Retourne une référence constante vers le nom complet de l'employé afin d'éviter toute copie inutile de la chaîne de caractères.

#### 4.7.3.4 getID()

```
double EmployeeIDCard::getID () const  [nodiscard]
```

Obtient le numéro d'identification de l'employé.

Obtient l'ID de l'employé.

**Returns**

Le numéro d'identification sous forme de `double`.

L'ID de l'employé sous forme de `double`.

Retourne l'ID de l'employé. L'ID est utilisé comme un `double`, mais il est généralement traité comme un entier dans ce contexte.

#### 4.7.3.5 getPosition()

```
const std::string & EmployeeIDCard::getPosition () const  [nodiscard]
```

Obtient le poste de l'employé.

Obtient le poste de l'employé dans l'entreprise.

**Returns**

Une référence constante vers le poste.

Une référence constante vers le poste de l'employé.

Retourne une référence constante vers le poste de l'employé, ce qui permet d'accéder à la donnée sans effectuer de copie.

### 4.7.4 Friends And Related Symbol Documentation

#### 4.7.4.1 HRManager

```
friend class HRManager  [friend]
```

Permet à la classe HRManager d'accéder aux membres privés.

#### 4.7.4.2 operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const EmployeeIDCard & IDCard)  [friend]
```

Surcharge de l'opérateur << pour afficher les informations de la carte d'identité.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie. |
| *IDCard* | La carte d'identité à afficher. |

**Returns**

Une référence vers le flux de sortie.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les informations de l'employé seront imprimées. |
| *IDCard* | La carte d'identité de l'employé à afficher. |

**Returns**

Une référence vers le flux de sortie.

Cette surcharge permet d'afficher les détails de la carte d'identité dans un format structuré et aligné.

#### 4.7.4.3 operator>>

```
std::istream & operator>> (
            std::istream & is,
            EmployeeIDCard & IDCard)  [friend]
```

Surcharge de l'opérateur >> pour lire les informations de la carte d'identité.

**Parameters**

| | |
|---|---|
| *is* | Le flux d'entrée. |
| *IDCard* | La carte d'identité à remplir. |

**Returns**

Une référence vers le flux d'entrée.

**Parameters**

| | |
|---|---|
| *is* | Le flux d'entrée où les informations seront lues. |
| *IDCard* | La carte d'identité de l'employé à remplir. |

**Returns**

Une référence vers le flux d'entrée.

Cette surcharge permet de saisir les informations de la carte d'identité de l'employé via le flux d'entrée, en demandant les différentes informations comme le nom, l'ID, etc.

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

- include/EmployeeIDCard.h
- src/EmployeeIDCard.cpp

## 4.8 Fleet Class Reference

Représente une flotte de véhicules.

`#include <Fleet.h>`

Collaboration diagram for Fleet:


### Public Member Functions

- Fleet ()=default

  *Constructeur par défaut de la flotte.*
- Fleet (std::vector< std::shared_ptr< Vehicule > > &&vehicules)

  *Constructeur paramétré pour initialiser la flotte avec une liste de véhicules.*
- std::shared_ptr< std::vector< std::shared_ptr< Vehicule > > > getVehicules () const

  *Obtient la liste des véhicules de la flotte.*
- std::vector< std::shared_ptr< Vehicule > > getBrand (const std::string &brand) const

  *Obtient tous les véhicules d'une marque spécifique.*
- std::shared_ptr< Vehicule > getVehicule (const std::string &brand, const std::string &model) const

  *Recherche un véhicule dans la flotte par sa marque et son modèle.*
- std::shared_ptr< Vehicule > getVehicule (const std::string &brand, const std::string &model, bool booked) const

  *Recherche un véhicule dans la flotte par sa marque, modèle et état de réservation.*
- std::shared_ptr< Vehicule > getVehicule (const std::string &brand, const std::string &model, unsigned int mileAge, bool booked=false) const

*Recherche un véhicule par sa marque, modèle et kilométrage.*

- bool hasVehicule (const std::shared_ptr< Vehicule > &vehicule) const

  *Vérifie si un véhicule spécifique est dans la flotte.*
- bool hasVehicule (const std::string &brand, const std::string &model) const

  *Vérifie si un véhicule avec une marque et un modèle spécifiques existe dans la flotte.*
- std::vector< std::shared_ptr< Vehicule > > getAvailableVehicules () const

  *Obtient la liste des véhicules disponibles dans la flotte (non réservés).*
- std::vector< std::shared_ptr< Vehicule > > getBookedVehicules () const

  *Obtient la liste des véhicules réservés dans la flotte.*
- void displayAvailableVehicules () const

  *Affiche les véhicules disponibles dans la flotte.*
- void displayBookedVehicules () const

  *Affiche les véhicules réservés dans la flotte.*
- void display () const

  *Affiche tous les véhicules de la flotte.*

**Friends**

- class FleetManager

### 4.8.1 Detailed Description

Représente une flotte de véhicules.

Cette classe gère une collection de véhicules (de type `Vehicule`) et offre des fonctionnalités pour accéder aux véhicules disponibles, réservés, et pour rechercher des véhicules par marque, modèle, kilométrage, etc.

### 4.8.2 Constructor & Destructor Documentation

#### 4.8.2.1 Fleet() [1/2]

```
Fleet::Fleet ()  [default]
```

Constructeur par défaut de la flotte.

#### 4.8.2.2 Fleet() [2/2]

```
Fleet::Fleet (
            std::vector< std::shared_ptr< Vehicule > > && vehicules)  [inline], [explicit]
```

Constructeur paramétré pour initialiser la flotte avec une liste de véhicules.

**Parameters**

| | |
|---|---|
| *vehicules* | La liste des véhicules à inclure dans la flotte. |

### 4.8.3 Member Function Documentation

#### 4.8.3.1 display()

```
void Fleet::display () const
```

Affiche tous les véhicules de la flotte.

Affiche le premier véhicule puis les autres véhicules de la flotte, qu'ils soient réservés ou non.

#### 4.8.3.2 displayAvailableVehicules()

```
void Fleet::displayAvailableVehicules () const
```

Affiche les véhicules disponibles dans la flotte.

La méthode parcourt la flotte et affiche les véhicules qui ne sont pas réservés.

#### 4.8.3.3 displayBookedVehicules()

```
void Fleet::displayBookedVehicules () const
```

Affiche les véhicules réservés dans la flotte.

La méthode parcourt la flotte et affiche les véhicules réservés.

#### 4.8.3.4 getAvailableVehicules()

```
std::vector< std::shared_ptr< Vehicule > > Fleet::getAvailableVehicules () const  [nodiscard]
```

Obtient la liste des véhicules disponibles dans la flotte (non réservés).

Obtient les véhicules disponibles dans la flotte.

**Returns**

Un vecteur contenant les véhicules disponibles.

Un vecteur contenant les véhicules disponibles (non réservés).

Cette méthode parcourt la flotte et retourne les véhicules qui ne sont pas réservés.

#### 4.8.3.5 getBookedVehicules()

```
std::vector< std::shared_ptr< Vehicule > > Fleet::getBookedVehicules () const  [nodiscard]
```

Obtient la liste des véhicules réservés dans la flotte.

Obtient les véhicules réservés dans la flotte.

**Returns**

Un vecteur contenant les véhicules réservés.

Un vecteur contenant les véhicules réservés.

Cette méthode parcourt la flotte et retourne les véhicules réservés.

#### 4.8.3.6 getBrand()

```
std::vector< std::shared_ptr< Vehicule > > Fleet::getBrand (
            const std::string & brand) const  [nodiscard]
```

Obtient tous les véhicules d'une marque spécifique.

**Parameters**

| | |
|---|---|
| *brand* | La marque des véhicules à rechercher. |

**Returns**

Un vecteur de pointeurs partagés vers les véhicules correspondants.

**Parameters**

| | |
|---|---|
| *brand* | La marque des véhicules à rechercher. |

**Returns**

Un vecteur de pointeurs partagés vers les véhicules de la marque spécifiée.

La méthode parcourt la flotte pour trouver tous les véhicules qui correspondent à la marque. Utilisation de `std::move()` pour éviter une copie coûteuse si un véhicule est trouvé.

#### 4.8.3.7 getVehicule() [1/3]

```
std::shared_ptr< Vehicule > Fleet::getVehicule (
            const std::string & brand,
            const std::string & model) const  [nodiscard]
```

Recherche un véhicule dans la flotte par sa marque et son modèle.

Obtient un véhicule spécifique par sa marque et son modèle.

**Parameters**

| | |
|---|---|
| *brand* | La marque du véhicule. |
| *model* | Le modèle du véhicule. |

**Returns**

Un pointeur partagé vers le véhicule trouvé, sinon nullptr.

**Parameters**

| | |
|---|---|
| *brand* | La marque du véhicule. |
| *model* | Le modèle du véhicule. |

**Returns**

Un pointeur partagé vers le véhicule trouvé.

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | Si aucun véhicule correspondant n'est trouvé. |

Cette méthode parcourt la flotte et retourne un `shared_ptr` vers le véhicule trouvé. En cas d'absence, une exception est lancée.

#### 4.8.3.8 getVehicule() [2/3]

```
std::shared_ptr< Vehicule > Fleet::getVehicule (
            const std::string & brand,
            const std::string & model,
            bool booked) const  [nodiscard]
```

Recherche un véhicule dans la flotte par sa marque, modèle et état de réservation.

Obtient un véhicule spécifique par sa marque, modèle et état de réservation.

**Parameters**

| | |
|---|---|
| *brand* | La marque du véhicule. |
| *model* | Le modèle du véhicule. |
| *booked* | L'état de réservation (true si réservé). |

**Returns**

Un pointeur partagé vers le véhicule trouvé, sinon nullptr.

**Parameters**

| | |
|---|---|
| *brand* | La marque du véhicule. |
| *model* | Le modèle du véhicule. |
| *booked* | L'état de réservation du véhicule (réservé ou non). |

**Returns**

Un pointeur partagé vers le véhicule trouvé.

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | Si aucun véhicule correspondant n'est trouvé. |

Cette méthode permet de filtrer les véhicules non seulement par marque et modèle, mais aussi par leur état de réservation.

#### 4.8.3.9 getVehicule() [3/3]

```
std::shared_ptr< Vehicule > Fleet::getVehicule (
            const std::string & brand,
            const std::string & model,
            unsigned int mileAge,
            bool booked = false) const  [nodiscard]
```

Recherche un véhicule par sa marque, modèle et kilométrage.

Obtient un véhicule spécifique par sa marque, modèle, kilométrage et état de réservation.

**Parameters**

| | |
|---|---|
| *brand* | La marque du véhicule. |
| *model* | Le modèle du véhicule. |
| *mileAge* | Le kilométrage du véhicule. |
| *booked* | L'état de réservation (par défaut false). |

**Returns**

Un pointeur partagé vers le véhicule trouvé, sinon nullptr.

**Parameters**

| | |
|---|---|
| *brand* | La marque du véhicule. |
| *model* | Le modèle du véhicule. |
| *mileAge* | Le kilométrage du véhicule. |
| *booked* | L'état de réservation du véhicule (réservé ou non). |

**Returns**

Un pointeur partagé vers le véhicule trouvé.

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | Si aucun véhicule correspondant n'est trouvé. |

Cette méthode permet de filtrer les véhicules par marque, modèle, kilométrage et état de réservation.

#### 4.8.3.10 getVehicules()

```
std::shared_ptr< std::vector< std::shared_ptr< Vehicule > > > Fleet::getVehicules () const
[nodiscard]
```

Obtient la liste des véhicules de la flotte.

**Returns**

Un pointeur partagé vers un vecteur de véhicules.

Un pointeur partagé vers un vecteur de pointeurs partagés vers les véhicules de la flotte.

Retourne la liste des véhicules sous forme de `shared_ptr` pour permettre une gestion partagée des véhicules sans effectuer de copies inutiles du vecteur.

#### 4.8.3.11 hasVehicule() [1/2]

```
bool Fleet::hasVehicule (
            const std::shared_ptr< Vehicule > & vehicule) const  [nodiscard]
```

Vérifie si un véhicule spécifique est dans la flotte.

Vérifie si un véhicule donné est dans la flotte.

**Parameters**

| | |
|---|---|
| *vehicule* | Le véhicule à vérifier. |

**Returns**

`true` si le véhicule est présent dans la flotte, sinon `false`.

**Parameters**

| | |
|---|---|
| *vehicule* | Le véhicule à vérifier. |

**Returns**

`true` si le véhicule est présent dans la flotte, sinon `false`.

Utilise `std::any_of` pour parcourir la flotte et vérifier si le véhicule est présent.

#### 4.8.3.12 hasVehicule() [2/2]

```
bool Fleet::hasVehicule (
            const std::string & brand,
            const std::string & model) const  [nodiscard]
```

Vérifie si un véhicule avec une marque et un modèle spécifiques existe dans la flotte.

Vérifie si un véhicule d'une marque et d'un modèle donnés existe dans la flotte.

**Parameters**

| | |
|---|---|
| *brand* | La marque du véhicule. |
| *model* | Le modèle du véhicule. |

**Returns**

`true` si le véhicule existe, sinon `false`.

**Parameters**

| | |
|---|---|
| *brand* | La marque du véhicule. |
| *model* | Le modèle du véhicule. |

**Returns**

`true` si le véhicule est présent, sinon `false`.

Utilise `std::any_of` pour vérifier si un véhicule avec les critères spécifiés existe dans la flotte.

### 4.8.4 Friends And Related Symbol Documentation

#### 4.8.4.1 FleetManager

```
friend class FleetManager  [friend]
```

Permet à la classe FleetManager d'accéder aux membres privés.

The documentation for this class was generated from the following files:

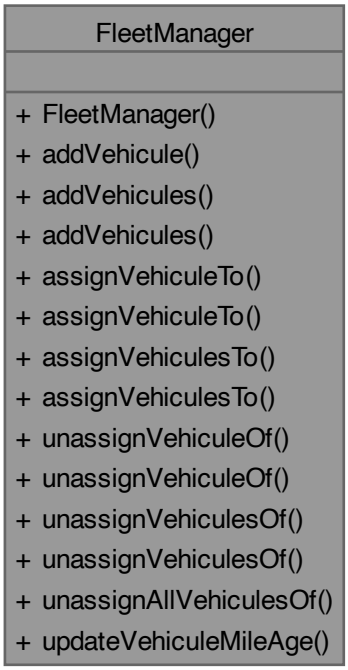
- include/Fleet.h
- src/Fleet.cpp

## 4.9 FleetManager Class Reference

Gère les véhicules de la flotte et leur assignation aux employés.

```
#include <FleetManager.h>
```

Collaboration diagram for FleetManager:



**Public Member Functions**

- FleetManager (EmployeeIDCard IDCard)

  *Constructeur paramétré pour initialiser le FleetManager avec une carte d'identité.*
- void addVehicule (Vehicule &vehicule)

  *Ajoute un véhicule à la flotte.*
- void addVehicules (std::vector< Vehicule > &&vehicules)

  *Ajoute plusieurs véhicules à la flotte.*
- void addVehicules (std::vector< ElectricVehicule > &&vehicules)

  *Ajoute plusieurs véhicules électriques à la flotte.*
- void assignVehiculeTo (std::shared_ptr< Vehicule > vehicule, const std::shared_ptr< Employee > &employee)

  *Assigne un véhicule à un employé.*
- void assignVehiculeTo (const std::string &brand, const std::string &model, const std::shared_ptr< Employee > &employee)

  *Assigne un véhicule à un employé par la marque et le modèle.*
- void assignVehiculesTo (const std::vector< std::array< std::string, 2 > > &brandModelCouples, const std::shared_ptr< Employee > &employee)

  *Assigne plusieurs véhicules à un employé par des paires marque-modèle.*
- void assignVehiculesTo (const std::vector< std::shared_ptr< Vehicule > > &vehicules, const std::shared_ptr< Employee > &employee)

  *Assigne plusieurs véhicules à un employé.*
- std::shared_ptr< Vehicule > unassignVehiculeOf (const std::shared_ptr< Vehicule > &vehicule, const std::shared_ptr< Employee > &employee)

  *Dissocie un véhicule d'un employé.*
- std::shared_ptr< Vehicule > unassignVehiculeOf (const std::string &brand, const std::string &model, const std::shared_ptr< Employee > &employee)

  *Dissocie un véhicule d'un employé par marque et modèle.*
- std::vector< std::shared_ptr< Vehicule > > unassignVehiculesOf (const std::vector< std::array< std::string, 2 > > &brandModelCouples, const std::shared_ptr< Employee > &employee)

  *Dissocie plusieurs véhicules d'un employé par des paires marque-modèle.*
- std::vector< std::shared_ptr< Vehicule > > unassignVehiculesOf (const std::vector< std::shared_ptr< Vehicule > > &vehicules, const std::shared_ptr< Employee > &employee)

  *Dissocie plusieurs véhicules d'un employé.*
- std::vector< std::shared_ptr< Vehicule > > unassignAllVehiculesOf (const std::shared_ptr< Employee > &employee)

  *Dissocie tous les véhicules d'un employé.*
- void updateVehiculeMileAge (const std::shared_ptr< Vehicule > &vehicule, unsigned int newMileAge)

  *Met à jour le kilométrage d'un véhicule.*

**Friends**

- class HRManager

### 4.9.1 Detailed Description

Gère les véhicules de la flotte et leur assignation aux employés.

Cette classe gère les opérations liées aux véhicules de la flotte, y compris l'ajout de véhicules, l'assignation de véhicules aux employés, et la mise à jour du kilométrage des véhicules. Elle permet également de dissocier les véhicules des employés.

## 4.9.2 Constructor & Destructor Documentation

### 4.9.2.1 FleetManager()

```
FleetManager::FleetManager (
            EmployeeIDCard IDCard)  [inline], [explicit]
```

Constructeur paramétré pour initialiser le FleetManager avec une carte d'identité.

**Parameters**

| | |
|---|---|
| *IDCard* | La carte d'identité de l'employé responsable de la gestion de la flotte. |

## 4.9.3 Member Function Documentation

### 4.9.3.1 addVehicule()

```
void FleetManager::addVehicule (
            Vehicule & vehicule)
```

Ajoute un véhicule à la flotte.

**Parameters**

| | |
|---|---|
| *vehicule* | Le véhicule à ajouter. |
| *vehicule* | Le véhicule à ajouter. |

Cette méthode utilise `dynamic_cast` pour vérifier si le véhicule est de type `ElectricVehicule`. Si c'est le cas, il est ajouté en tant qu'objet `ElectricVehicule`, sinon il est ajouté comme un objet `Vehicule` standard.

### 4.9.3.2 addVehicules() [1/2]

```
void FleetManager::addVehicules (
            std::vector< ElectricVehicule > && vehicules)
```

Ajoute plusieurs véhicules électriques à la flotte.

**Parameters**

| | |
|---|---|
| *vehicules* | La liste des véhicules électriques à ajouter. |
| *vehicules* | La liste des véhicules électriques à ajouter. |

Semblable à `addVehicules(std::vector<Vehicule>&&)`, mais cette méthode est spécifique aux véhicules électriques. Elle itère sur la liste des véhicules électriques et les ajoute à la flotte.

### 4.9.3.3 addVehicules() [2/2]

```
void FleetManager::addVehicules (
            std::vector< Vehicule > && vehicules)
```

Ajoute plusieurs véhicules à la flotte.

**Parameters**

| | |
|---|---|
| *vehicules* | La liste des véhicules à ajouter à la flotte. |
| *vehicules* | La liste des véhicules à ajouter. |

Cette méthode itère sur un vecteur de véhicules et appelle `addVehicule()` pour ajouter chaque véhicule individuellement à la flotte.

#### 4.9.3.4 assignVehiculesTo() [1/2]

```
void FleetManager::assignVehiculesTo (
            const std::vector< std::array< std::string, 2 > > & brandModelCouples,
            const std::shared_ptr< Employee > & employee)
```

Assigne plusieurs véhicules à un employé par des paires marque-modèle.

Assigne plusieurs véhicules à un employé en utilisant une liste de paires marque-modèle.

**Parameters**

| | |
|---|---|
| *brandModelCouples* | Une liste de couples marque-modèle. |
| *employee* | L'employé à qui les véhicules sont assignés. |
| *brandModelCouples* | Une liste de couples marque-modèle. |
| *employee* | L'employé à qui les véhicules seront assignés. |

Cette méthode assigne les véhicules à l'employé en fonction des couples marque-modèle.

#### 4.9.3.5 assignVehiculesTo() [2/2]

```
void FleetManager::assignVehiculesTo (
            const std::vector< std::shared_ptr< Vehicule > > & vehicules,
            const std::shared_ptr< Employee > & employee)
```

Assigne plusieurs véhicules à un employé.

**Parameters**

| | |
|---|---|
| *vehicules* | La liste des véhicules à assigner. |
| *employee* | L'employé à qui les véhicules sont assignés. |
| *vehicules* | La liste des véhicules à assigner. |
| *employee* | L'employé à qui les véhicules seront assignés. |

Cette méthode itère sur la liste des véhicules et les assigne un par un à l'employé.

#### 4.9.3.6 assignVehiculeTo() [1/2]

```
void FleetManager::assignVehiculeTo (
            const std::string & brand,
            const std::string & model,
            const std::shared_ptr< Employee > & employee)
```

Assigne un véhicule à un employé par la marque et le modèle.

Assigne un véhicule à un employé en utilisant la marque et le modèle.

**Parameters**

| | |
|---|---|
| *brand* | La marque du véhicule. |
| *model* | Le modèle du véhicule. |
| *employee* | L'employé à qui le véhicule est assigné. |
| *brand* | La marque du véhicule. |
| *model* | Le modèle du véhicule. |
| *employee* | L'employé à qui le véhicule sera assigné. |

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | Si le véhicule est déjà réservé. |

Utilise `fleet->getVehicule()` pour récupérer le véhicule et vérifier s'il est réservé avant de l'assigner à l'employé.

#### 4.9.3.7 assignVehiculeTo() [2/2]

```
void FleetManager::assignVehiculeTo (
            std::shared_ptr< Vehicule > vehicule,
            const std::shared_ptr< Employee > & employee)
```

Assigne un véhicule à un employé.

**Parameters**

| | |
|---|---|
| *vehicule* | Le véhicule à assigner. |
| *employee* | L'employé à qui le véhicule est assigné. |
| *vehicule* | Le véhicule à assigner. |
| *employee* | L'employé à qui le véhicule sera assigné. |

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | Si le véhicule n'est pas dans la flotte ou s'il est déjà réservé. |

Cette méthode vérifie d'abord que le véhicule est dans la flotte et qu'il n'est pas déjà réservé. Si ces conditions sont remplies, elle marque le véhicule comme réservé et l'ajoute à la liste des véhicules réservés de l'employé.

#### 4.9.3.8 unassignAllVehiculesOf()

```
std::vector< std::shared_ptr< Vehicule > > FleetManager::unassignAllVehiculesOf (
            const std::shared_ptr< Employee > & employee)
```

Dissocie tous les véhicules d'un employé.

Dissocie tous les véhicules réservés d'un employé.

**Parameters**

| | |
|---|---|
| *employee* | L'employé duquel tous les véhicules sont dissociés. |

**Returns**

Une liste des véhicules dissociés.

**Parameters**

| | |
|---|---|
| *employee* | L'employé dont tous les véhicules réservés seront dissociés. |

**Returns**

Une liste des véhicules dissociés.

Cette méthode parcourt la liste des véhicules réservés par l'employé, marque chaque véhicule comme non réservé (`booked = false`), et les ajoute à un vecteur avant de les retirer de la liste de l'employé. La méthode renvoie la liste des véhicules dissociés.

#### 4.9.3.9 unassignVehiculeOf() [1/2]

```
std::shared_ptr< Vehicule > FleetManager::unassignVehiculeOf (
            const std::shared_ptr< Vehicule > & vehicule,
            const std::shared_ptr< Employee > & employee)
```

Dissocie un véhicule d'un employé.

**Parameters**

| | |
|---|---|
| *vehicule* | Le véhicule à dissocier. |
| *employee* | L'employé duquel le véhicule est dissocié. |

**Returns**

Un pointeur partagé vers le véhicule dissocié.

**Parameters**

| | |
|---|---|
| *vehicule* | Le véhicule à dissocier. |
| *employee* | L'employé dont le véhicule sera dissocié. |

**Returns**

Le véhicule dissocié.

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | Si le véhicule n'est pas réservé par l'employé. |

Cette méthode vérifie que l'employé a réservé le véhicule avant de le dissocier et de le renvoyer.

#### 4.9.3.10 unassignVehiculeOf() [2/2]

```
std::shared_ptr< Vehicule > FleetManager::unassignVehiculeOf (
            const std::string & brand,
            const std::string & model,
            const std::shared_ptr< Employee > & employee)
```

Dissocie un véhicule d'un employé par marque et modèle.

Dissocie un véhicule d'un employé en utilisant la marque et le modèle.

**Parameters**

| | |
|---|---|
| *brand* | La marque du véhicule. |
| *model* | Le modèle du véhicule. |
| *employee* | L'employé duquel le véhicule est dissocié. |

**Returns**

Un pointeur partagé vers le véhicule dissocié.

**Parameters**

| | |
|---|---|
| *brand* | La marque du véhicule. |
| *model* | Le modèle du véhicule. |
| *employee* | L'employé dont le véhicule sera dissocié. |

**Returns**

Le véhicule dissocié.

Cette méthode appelle `unassignVehiculeOf` en utilisant la marque et le modèle pour identifier le véhicule à dissocier.

#### 4.9.3.11 unassignVehiculesOf() [1/2]

```
std::vector< std::shared_ptr< Vehicule > > FleetManager::unassignVehiculesOf (
            const std::vector< std::array< std::string, 2 > > & brandModelCouples,
            const std::shared_ptr< Employee > & employee)
```

Dissocie plusieurs véhicules d'un employé par des paires marque-modèle.

Dissocie plusieurs véhicules d'un employé en utilisant des couples marque-modèle.

**Parameters**

| | |
|---|---|
| *brandModelCouples* | Une liste de couples marque-modèle. |
| *employee* | L'employé duquel les véhicules sont dissociés. |

**Returns**

Une liste des véhicules dissociés.

**Parameters**

| | |
|---|---|
| *brandModelCouples* | Une liste de couples marque-modèle. |
| *employee* | L'employé dont les véhicules seront dissociés. |

**Returns**

Une liste des véhicules dissociés.

Cette méthode dissocie plusieurs véhicules d'un employé en fonction des couples marque-modèle.

#### 4.9.3.12 unassignVehiculesOf() [2/2]

```
std::vector< std::shared_ptr< Vehicule > > FleetManager::unassignVehiculesOf (
            const std::vector< std::shared_ptr< Vehicule > > & vehicules,
            const std::shared_ptr< Employee > & employee)
```

Dissocie plusieurs véhicules d'un employé.

**Parameters**

| | |
|---|---|
| *vehicules* | La liste des véhicules à dissocier. |
| *employee* | L'employé duquel les véhicules sont dissociés. |

**Returns**

Une liste des véhicules dissociés.

**Parameters**

| | |
|---|---|
| *vehicules* | La liste des véhicules à dissocier. |
| *employee* | L'employé dont les véhicules seront dissociés. |

**Returns**

Une liste des véhicules dissociés.

Cette méthode dissocie tous les véhicules dans la liste spécifiée de l'employé.

#### 4.9.3.13 updateVehiculeMileAge()

```
void FleetManager::updateVehiculeMileAge (
            const std::shared_ptr< Vehicule > & vehicule,
            unsigned int newMileAge)
```

Met à jour le kilométrage d'un véhicule.

**Parameters**

| | |
|---|---|
| *vehicule* | Le véhicule à mettre à jour. |
| *newMileAge* | Le nouveau kilométrage du véhicule. |
| *vehicule* | Le véhicule dont le kilométrage sera mis à jour. |
| *newMileAge* | Le nouveau kilométrage à attribuer au véhicule. |

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | Si le véhicule n'existe pas dans la flotte. |

Cette méthode vérifie d'abord si le véhicule est bien dans la flotte. Si c'est le cas, elle met à jour le kilométrage du véhicule. Si le véhicule n'est pas trouvé dans la flotte, une exception est levée.

### 4.9.4 Friends And Related Symbol Documentation

#### 4.9.4.1 HRManager

```
friend class HRManager  [friend]
```

Permet à la classe HRManager d'accéder aux membres privés.

The documentation for this class was generated from the following files:

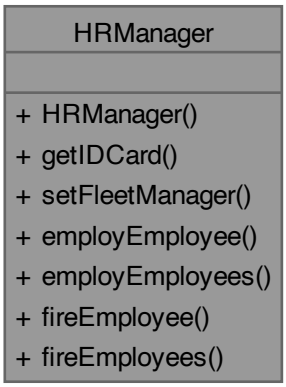
- include/FleetManager.h
- src/FleetManager.cpp

## 4.10 HRManager Class Reference

Gère les employés d'une entreprise, y compris l'embauche, le licenciement et l'affectation du gestionnaire de flotte.

```
#include <HRManager.h>
```

Collaboration diagram for HRManager:



**Public Member Functions**

- HRManager (EmployeeIDCard IDCard)

  *Constructeur paramétré pour initialiser le gestionnaire RH avec une carte d'identité.*
- const EmployeeIDCard & getIDCard () const

  *Obtient la carte d'identité de l'employé.*
- void setFleetManager (FleetManager &&fleetManager)

  *Assigne un gestionnaire de flotte à l'entreprise.*
- void employEmployee (Employee &&employee)

  *Embauche un employé pour l'entreprise.*
- void employEmployees (std::vector< Employee > &&employees)

  *Embauche plusieurs employés pour l'entreprise.*
- std::shared_ptr< Employee > fireEmployee (const std::shared_ptr< Employee > &employee)

  *Licencie un employé de l'entreprise.*
- std::vector< std::shared_ptr< Employee > > fireEmployees (std::vector< std::shared_ptr< Employee > > &&employees)

  *Licencie plusieurs employés de l'entreprise.*

**Friends**

- class CEO

### 4.10.1 Detailed Description

Gère les employés d'une entreprise, y compris l'embauche, le licenciement et l'affectation du gestionnaire de flotte.

La classe HRManager permet de gérer les ressources humaines d'une entreprise, y compris l'affectation des gestionnaires de flotte, l'embauche et le licenciement des employés.

### 4.10.2 Constructor & Destructor Documentation

#### 4.10.2.1 HRManager()

```
HRManager::HRManager (
            EmployeeIDCard IDCard)  [inline], [explicit]
```

Constructeur paramétré pour initialiser le gestionnaire RH avec une carte d'identité.

**Parameters**

| | |
|---|---|
| *IDCard* | La carte d'identité de l'employé responsable des ressources humaines. |

### 4.10.3 Member Function Documentation

#### 4.10.3.1 employEmployee()

```
void HRManager::employEmployee (
            Employee && employee)
```

Embauche un employé pour l'entreprise.

**Parameters**

| | |
|---|---|
| *employee* | L'employé à embaucher. |
| *employee* | L'employé à embaucher. |

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | Si le nom de l'entreprise de l'employé ne correspond pas à celui de l'entreprise. |
| *std::invalid_argument* | Si un employé avec le même ID est déjà employé. |

La méthode vérifie que l'employé a un nom d'entreprise qui correspond à celui de l'entreprise. Elle s'assure également que l'employé n'existe pas déjà dans la liste des employés de l'entreprise avant de l'ajouter.

#### 4.10.3.2 employEmployees()

```
void HRManager::employEmployees (
            std::vector< Employee > && employees)
```

Embauche plusieurs employés pour l'entreprise.

**Parameters**

| | |
|---|---|
| *employees* | La liste des employés à embaucher. |
| *employees* | La liste des employés à embaucher. |

Appelle `employEmployee()` pour chaque employé dans le vecteur. Utilise `std::move()` pour éviter de faire des copies de chaque employé.

#### 4.10.3.3 fireEmployee()

```
std::shared_ptr< Employee > HRManager::fireEmployee (
            const std::shared_ptr< Employee > & employee)
```

Licencie un employé de l'entreprise.

**Parameters**

| | |
|---|---|
| *employee* | L'employé à licencier. |

**Returns**

Un pointeur partagé vers l'employé licencié.

**Parameters**

| | |
|---|---|
| *employee* | L'employé à licencier. |

**Returns**

L'employé licencié.

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | Si l'employé n'est pas trouvé dans la liste des employés de l'entreprise. |

Cette méthode recherche l'employé dans la liste des employés, dissocie tous les véhicules de l'employé, et le retire de la liste. Elle renvoie l'employé licencié.

#### 4.10.3.4 fireEmployees()

```
std::vector< std::shared_ptr< Employee > > HRManager::fireEmployees (
            std::vector< std::shared_ptr< Employee > > && employees)
```

Licencie plusieurs employés de l'entreprise.

**Parameters**

| | |
|---|---|
| *employees* | La liste des employés à licencier. |

**Returns**

Un vecteur de pointeurs partagés vers les employés licenciés.

**Parameters**

| | |
|---|---|
| *employees* | La liste des employés à licencier. |

**Returns**

Une liste des employés licenciés.

Cette méthode appelle `fireEmployee()` pour chaque employé et renvoie une liste des employés licenciés.

#### 4.10.3.5 getIDCard()

```
const EmployeeIDCard & HRManager::getIDCard () const  [nodiscard]
```

Obtient la carte d'identité de l'employé.

Obtient la carte d'identité de l'employé responsable des ressources humaines.

**Returns**

Une référence constante vers l'objet EmployeeIDCard.

Une référence constante vers la carte d'identité de l'employé RH.

Retourne la carte d'identité de l'employé RH par référence constante pour éviter une copie inutile, car la carte d'identité ne change pas dans cette méthode.

#### 4.10.3.6 setFleetManager()

```
void HRManager::setFleetManager (
            FleetManager && fleetManager)
```

Assigne un gestionnaire de flotte à l'entreprise.

**Parameters**

| | |
|---|---|
| *fleetManager* | Le gestionnaire de flotte à affecter. |
| *fleetManager* | Le gestionnaire de flotte à affecter. |

Déplace un `FleetManager` existant dans l'entreprise en l'assignant à la société. La méthode utilise `std::move()` pour éviter une copie et permet le transfert de la gestion du gestionnaire de flotte.

### 4.10.4 Friends And Related Symbol Documentation

#### 4.10.4.1 CEO

```
friend class CEO  [friend]
```

Permet à la classe CEO d'accéder aux membres privés.

The documentation for this class was generated from the following files:

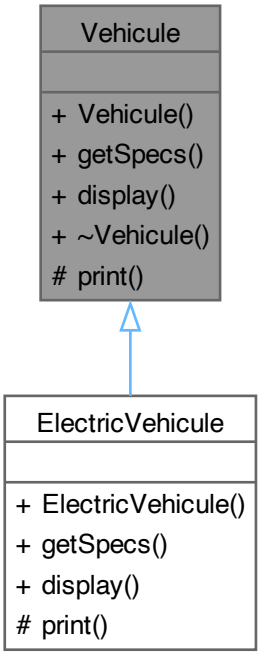
- include/HRManager.h
- src/HRManager.cpp

## 4.11 Vehicule Class Reference

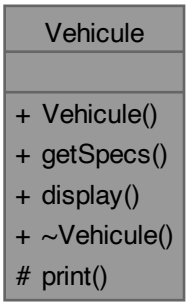
Représente un véhicule avec des spécifications.

```
#include <Vehicule.h>
```

Inheritance diagram for Vehicule:



Collaboration diagram for Vehicule:




#### Public Member Functions

- Vehicule (VehiculeSpecs specs)

  *Constructeur paramétré pour initialiser un véhicule avec ses spécifications.*
- virtual const VehiculeSpecs & getSpecs ()

  *Obtient les spécifications du véhicule.*
- virtual void display () const

  *Affiche les informations du véhicule.*
- virtual ~Vehicule ()=default

  *Destructeur virtuel pour garantir la destruction correcte des objets dérivés.*

#### Protected Member Functions

- virtual void print (std::ostream &os) const

  *Méthode protégée pour imprimer les informations du véhicule dans un flux de sortie.*

#### Friends

- class FleetManager
- std::ostream & operator<< (std::ostream &os, const Vehicule &vehicule)

  *Surcharge de l'opérateur << pour afficher un véhicule.*

### 4.11.1 Detailed Description

Représente un véhicule avec des spécifications.

La classe `Vehicule` est une classe de base qui contient des informations communes à tous les véhicules, telles que les spécifications du véhicule (`VehiculeSpecs`). Elle fournit également des méthodes pour afficher et manipuler ces spécifications.

### 4.11.2 Constructor & Destructor Documentation

#### 4.11.2.1 Vehicule()

```
Vehicule::Vehicule (
            VehiculeSpecs specs)  [inline], [explicit]
```

Constructeur paramétré pour initialiser un véhicule avec ses spécifications.

**Parameters**

| | |
|---|---|
| *specs* | Les spécifications du véhicule. |

#### 4.11.2.2 ~Vehicule()

```
virtual Vehicule::~Vehicule ()  [virtual], [default]
```

Destructeur virtuel pour garantir la destruction correcte des objets dérivés.

### 4.11.3 Member Function Documentation

#### 4.11.3.1 display()

```
void Vehicule::display () const  [virtual]
```

Affiche les informations du véhicule.

Affiche les spécifications du véhicule.

Cette méthode est virtuelle et peut être redéfinie dans les classes dérivées pour afficher des informations spécifiques au type de véhicule.

Appelle la méthode `display()` des spécifications pour afficher les informations détaillées du véhicule de manière lisible.

Reimplemented in ElectricVehicule.

#### 4.11.3.2 getSpecs()

```
const VehiculeSpecs & Vehicule::getSpecs ()  [nodiscard], [virtual]
```

Obtient les spécifications du véhicule.

**Returns**

Une référence constante vers les spécifications du véhicule.

Une référence constante vers les spécifications du véhicule.

Retourne les spécifications du véhicule par référence pour éviter la copie et permettre un accès efficace aux données.

Reimplemented in ElectricVehicule.

#### 4.11.3.3 print()

```
void Vehicule::print (
            std::ostream & os) const  [protected], [virtual]
```

Méthode protégée pour imprimer les informations du véhicule dans un flux de sortie.

Imprime les spécifications du véhicule dans un flux de sortie.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les informations seront imprimées. |
| *os* | Le flux de sortie où les spécifications du véhicule seront imprimées. |

Cette méthode appelle l'opérateur << pour afficher les spécifications du véhicule. Les spécifications sont imprimées sans effectuer de copie du véhicule.

Reimplemented in ElectricVehicule.

## 4.11.4 Friends And Related Symbol Documentation

### 4.11.4.1 FleetManager

```
friend class FleetManager  [friend]
```

Permet à la classe FleetManager d'accéder aux membres privés.

### 4.11.4.2 operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const Vehicule & vehicule)  [friend]
```

Surcharge de l'opérateur << pour afficher un véhicule.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les informations du véhicule seront imprimées. |
| *vehicule* | Le véhicule à afficher. |

**Returns**

Une référence vers le flux de sortie.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les informations du véhicule seront imprimées. |
| *vehicule* | Le véhicule à afficher. |

**Returns**

Une référence vers le flux de sortie.

L'opérateur << est utilisé pour afficher un véhicule en appelant la méthode `print()`. Cela permet un affichage structuré des informations du véhicule sans effectuer de copie.

The documentation for this class was generated from the following files:

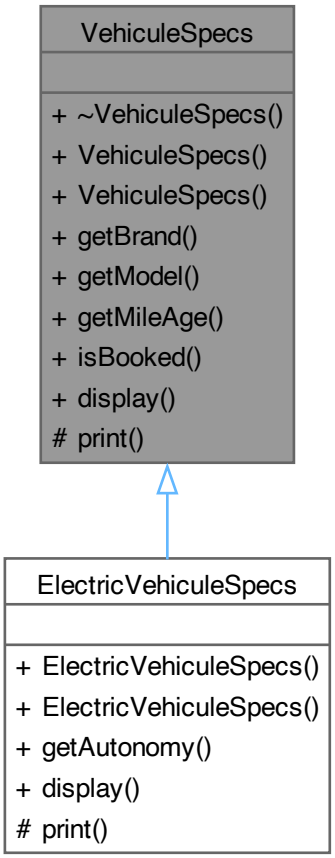
- include/Vehicule.h
- src/Vehicule.cpp

## 4.12 VehiculeSpecs Struct Reference

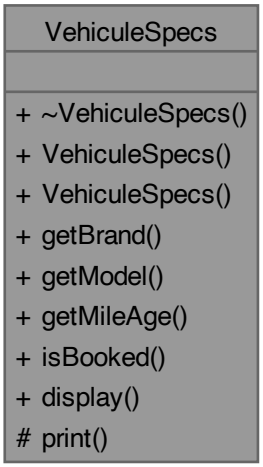
Représente les spécifications de base d'un véhicule.

`#include <VehiculeSpecs.h>`

Inheritance diagram for VehiculeSpecs:



Collaboration diagram for VehiculeSpecs:




### Public Member Functions

- virtual ~VehiculeSpecs ()=default

  *Destructeur virtuel pour permettre la destruction correcte des objets dérivés.*
- VehiculeSpecs (std::string brand, std::string model)

  *Constructeur pour initialiser la marque et le modèle du véhicule.*
- VehiculeSpecs (std::string brand, std::string model, unsigned int mileAge, bool booked)

  *Constructeur pour initialiser toutes les spécifications du véhicule.*
- const std::string & getBrand () const

  *Obtient la marque du véhicule.*
- const std::string & getModel () const

  *Obtient le modèle du véhicule.*
- unsigned int getMileAge () const

  *Obtient le kilométrage du véhicule.*
- bool isBooked () const

  *Vérifie si le véhicule est réservé.*
- virtual void display () const

  *Affiche les spécifications du véhicule.*

### Protected Member Functions

- virtual void print (std::ostream &os) const

  *Méthode protégée pour imprimer les spécifications du véhicule dans un flux de sortie.*

### Friends

- class FleetManager
- std::ostream & operator<< (std::ostream &os, const VehiculeSpecs &specs)

  *Surcharge de l'opérateur << pour afficher les spécifications d'un véhicule.*

### 4.12.1 Detailed Description

Représente les spécifications de base d'un véhicule.

Cette structure contient des informations communes à tous les véhicules, telles que la marque, le modèle, le kilométrage et l'état de réservation. Elle sert de base pour des classes dérivées spécifiques aux types de véhicules.

### 4.12.2 Constructor & Destructor Documentation

#### 4.12.2.1 ~VehiculeSpecs()

```
virtual VehiculeSpecs::~VehiculeSpecs ()  [virtual], [default]
```

Destructeur virtuel pour permettre la destruction correcte des objets dérivés.

#### 4.12.2.2 VehiculeSpecs() [1/2]

```
VehiculeSpecs::VehiculeSpecs (
            std::string brand,
            std::string model)  [inline]
```

Constructeur pour initialiser la marque et le modèle du véhicule.

**Parameters**

| | |
|---|---|
| *brand* | La marque du véhicule. |
| *model* | Le modèle du véhicule. |

#### 4.12.2.3 VehiculeSpecs() [2/2]

```
VehiculeSpecs::VehiculeSpecs (
            std::string brand,
            std::string model,
            unsigned int mileAge,
            bool booked)  [inline]
```

Constructeur pour initialiser toutes les spécifications du véhicule.

**Parameters**

| | |
|---|---|
| *brand* | La marque du véhicule. |
| *model* | Le modèle du véhicule. |
| *mileAge* | Le kilométrage du véhicule. |
| *booked* | L'état de réservation du véhicule (true si réservé). |

### 4.12.3 Member Function Documentation

#### 4.12.3.1 display()

```
void VehiculeSpecs::display () const  [virtual]
```

Affiche les spécifications du véhicule.

Affiche les spécifications du véhicule de manière lisible.

Cette méthode est virtuelle et peut être redéfinie dans les classes dérivées pour afficher des informations spécifiques au type de véhicule.

Cette méthode affiche un en-tête avec les titres des colonnes et utilise l'opérateur $<<$ pour afficher les informations détaillées du véhicule.

Reimplemented in ElectricVehiculeSpecs.

#### 4.12.3.2 getBrand()

```
const std::string & VehiculeSpecs::getBrand () const  [nodiscard]
```

Obtient la marque du véhicule.

**Returns**

La marque du véhicule.

Une référence constante vers la marque du véhicule.

Retourne la marque du véhicule par référence pour éviter la copie inutile.

#### 4.12.3.3 getMileAge()

```
unsigned int VehiculeSpecs::getMileAge () const  [nodiscard]
```

Obtient le kilométrage du véhicule.

**Returns**

Le kilométrage du véhicule.

Le kilométrage du véhicule sous forme d'un entier non signé.

Retourne le kilométrage du véhicule directement sans le copier.

#### 4.12.3.4 getModel()

```
const std::string & VehiculeSpecs::getModel () const  [nodiscard]
```

Obtient le modèle du véhicule.

**Returns**

Le modèle du véhicule.

Une référence constante vers le modèle du véhicule.

Retourne le modèle du véhicule par référence pour éviter la copie inutile.

#### 4.12.3.5 isBooked()

```
bool VehiculeSpecs::isBooked () const  [nodiscard]
```

Vérifie si le véhicule est réservé.

**Returns**

`true` si le véhicule est réservé, sinon `false`.

`true` si le véhicule est réservé, sinon `false`.

Retourne l'état de réservation du véhicule, ce qui permet de savoir si le véhicule est actuellement réservé.

#### 4.12.3.6 print()

```
void VehiculeSpecs::print (
            std::ostream & os) const  [protected], [virtual]
```

Méthode protégée pour imprimer les spécifications du véhicule dans un flux de sortie.

Imprime les spécifications du véhicule dans un flux de sortie.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les informations seront imprimées. |
| *os* | Le flux de sortie où les spécifications seront imprimées. |

Cette méthode utilise `std::setw` pour formater l'affichage des spécifications du véhicule, en affichant la marque, le modèle, le kilométrage, l'état de réservation et l'autonomie.

Reimplemented in ElectricVehiculeSpecs.

### 4.12.4 Friends And Related Symbol Documentation

#### 4.12.4.1 FleetManager

```
friend class FleetManager  [friend]
```

Permet à la classe FleetManager d'accéder aux membres privés.

#### 4.12.4.2 operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const VehiculeSpecs & specs)  [friend]
```

Surcharge de l'opérateur << pour afficher les spécifications d'un véhicule.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie. |
| *specs* | Les spécifications du véhicule à afficher. |

**Returns**

Une référence vers le flux de sortie.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les spécifications seront imprimées. |
| *specs* | Les spécifications du véhicule à afficher. |

**Returns**

Une référence vers le flux de sortie.

L'opérateur << est utilisé pour afficher les spécifications du véhicule de manière structurée, en appelant la méthode `print()`.

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

- include/VehiculeSpecs.h
- src/VehiculeSpecs.cpp

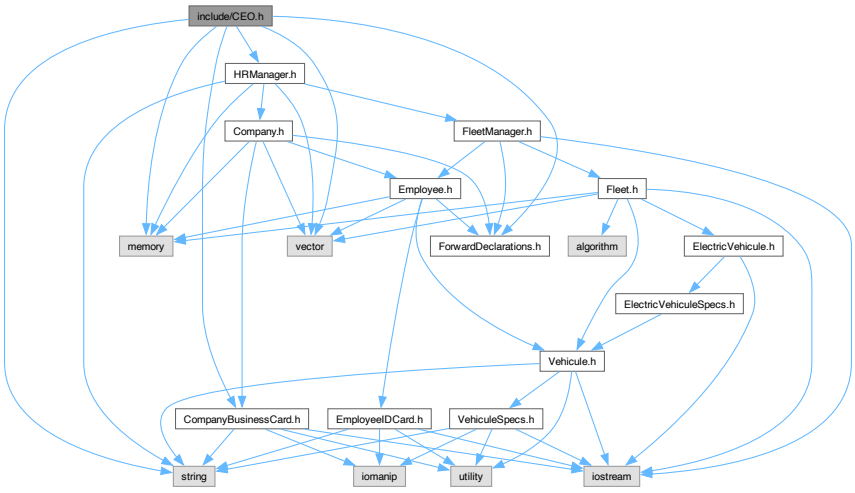
# Chapter 5

# File Documentation

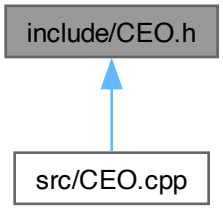
## 5.1 include/CEO.h File Reference

```
#include <string>
#include <memory>
#include <vector>
#include "ForwardDeclarations.h"
#include "CompanyBusinessCard.h"
#include "HRManager.h"
```

Include dependency graph for CEO.h:



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




### Classes

- class CEO

  *Représente un PDG (Chief Executive Officer) qui gère plusieurs entreprises.*

## 5.2 CEO.h

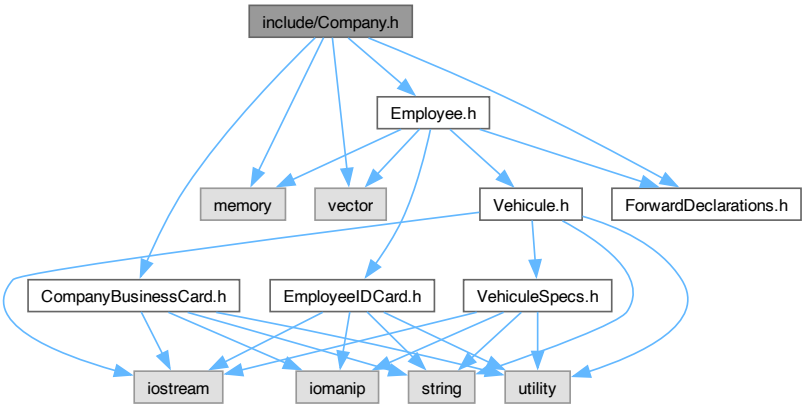
Go to the documentation of this file.

```
00001 #pragma once
00002 #include <string>
00003 #include <memory>
00004 #include <vector>
00005 #include "ForwardDeclarations.h"
00006 #include "CompanyBusinessCard.h"
00007 #include "HRManager.h"
00008
00016 class CEO
00017 {
00018 private:
00019     std::string fullName;
00020     std::vector<std::shared_ptr<Company» companies;
00022 public:
00024     CEO() = default;
00025
00030     explicit CEO(std::string fullName)
00031         : fullName(std::move(fullName))
00032     {}
00033
00038     [[nodiscard]] const std::string& getFullName() const;
00039
00044     void launchCompany(CompanyBusinessCard IDCard);
00045
00051     [[nodiscard]] std::shared_ptr<Company> getCompany(
00052         const std::string& name
00053     ) const;
00054
00059     void setFullName(std::string fullName);
00060
00066     void setCompanyBusinessCard(const std::string& companyName, CompanyBusinessCard businessCard);
00067
00073     void setHRManagerFor(HRManager&& managerOfHR, const std::string& name);
00074
00080     void makeFleetFor(Fleet&& fleet, const std::string& name);
00081
00086     void shutDownCompany(const std::string& name);
00087
00094     friend std::istream& operator»(std::istream& is, CEO& ceo);
00095 };
```

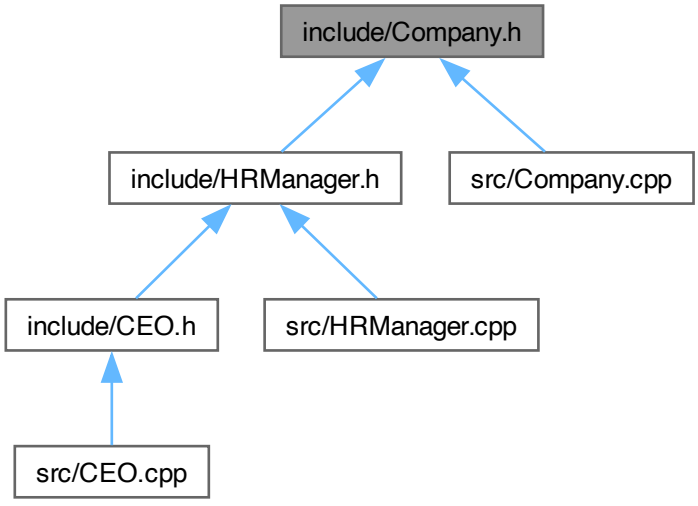
## 5.3 include/Company.h File Reference

```
#include <memory>
#include <vector>
#include "ForwardDeclarations.h"
#include "CompanyBusinessCard.h"
#include "Employee.h"
```

Include dependency graph for Company.h:




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




### Classes

- class Company

  *Représente une entreprise avec ses employés, sa flotte de véhicules, et son gestionnaire RH.*

## 5.4 Company.h

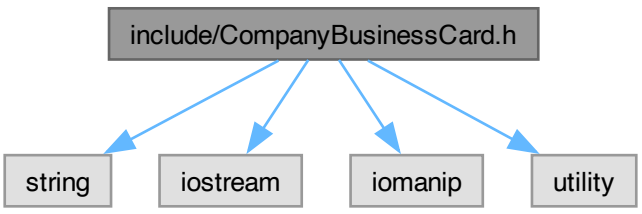
Go to the documentation of this file.

```
00001 #pragma once
00002 #include <memory>
00003 #include <vector>
00004 #include "ForwardDeclarations.h"
00005 #include "CompanyBusinessCard.h"
00006 #include "Employee.h"
00007
00008 class CEO;
00009
00018 class Company
00019 {
00020 private:
00021     friend class CEO;
00022     friend class HRManager;
00024     CompanyBusinessCard businessCard;
00025     std::shared_ptr<HRManager> managerOfHR;
00026     std::shared_ptr<std::vector<std::shared_ptr<Employee»> employees =
00027         std::make_shared<std::vector<std::shared_ptr<Employee»>();
00028     std::shared_ptr<FleetManager> fleetManager;
00029     std::shared_ptr<Fleet> fleet;
00031 public:
00036     explicit Company(CompanyBusinessCard businessCard) : businessCard(std::move(businessCard))
00037     {}
00038
00043     [[nodiscard]] const CompanyBusinessCard& getBusinessCard() const;
00044
00049     [[nodiscard]] std::shared_ptr<HRManager> getHRManager() const;
00050
00055     [[nodiscard]] std::shared_ptr<FleetManager> getFleetManager() const;
00056
00061     [[nodiscard]] std::shared_ptr<Fleet> getFleet() const;
00062
00067     [[nodiscard]] std::shared_ptr<std::vector<std::shared_ptr<Employee»> getEmployees() const;
00068
00073     [[nodiscard]] size_t employeesCount() const;
00074
00078     void displayEmployees() const;
00079 };
```

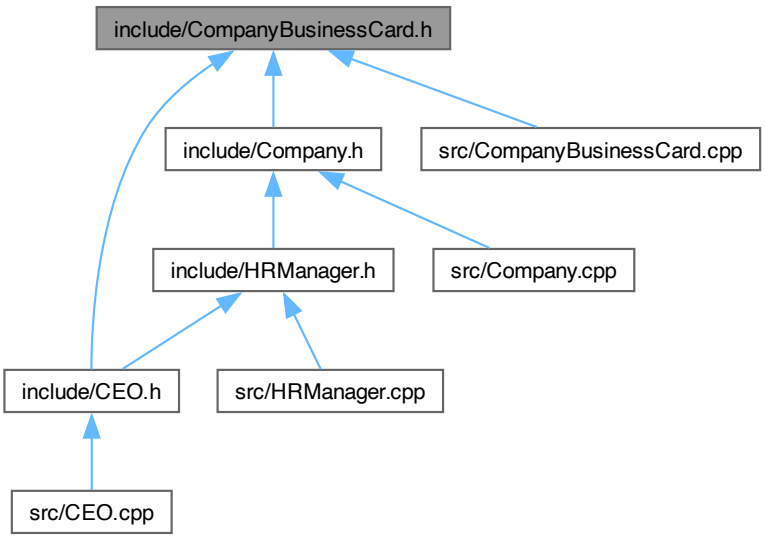
## 5.5 include/CompanyBusinessCard.h File Reference

```
#include <string>
#include <iostream>
#include <iomanip>
#include <utility>
```

Include dependency graph for CompanyBusinessCard.h:



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




### Classes

- struct CompanyBusinessCard

  *Représente une carte professionnelle pour une entreprise.*

## 5.6 CompanyBusinessCard.h

Go to the documentation of this file.

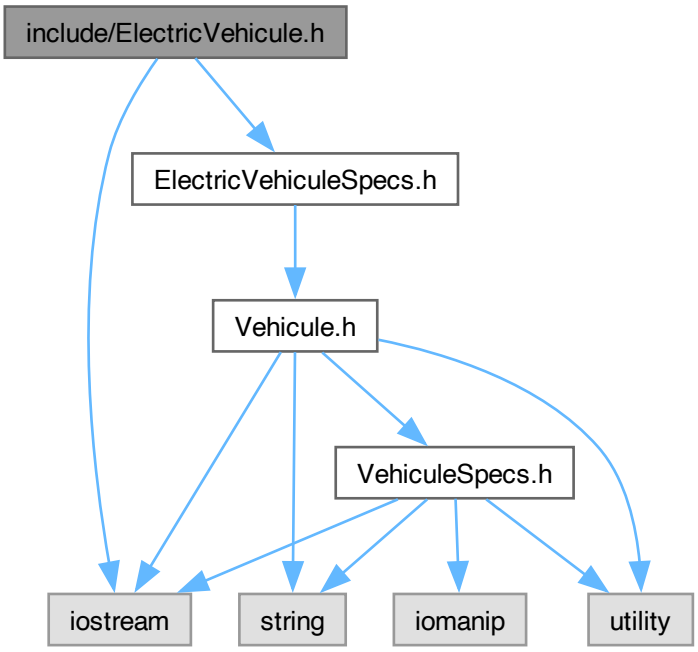
```
00001 #pragma once
00002 #include <string>
00003 #include <iostream>
00004 #include <iomanip>
00005 #include <utility>
00006
00014 struct CompanyBusinessCard
00015 {
00016 private:
00017     std::string name;
00018     std::string address;
00019     std::string phoneNumber;
00020     std::string email;
00022 public:
00024     CompanyBusinessCard() = default;
00025
00033     CompanyBusinessCard(
00034         std::string name,
00035         std::string address,
00036         std::string phoneNumber,
00037         std::string email
00038     ) : name(std::move(name)),
00039         address(std::move(address)),
00040         phoneNumber(std::move(phoneNumber)),
00041         email(std::move(email))
00042     {
00043     }
00044
00049     [[nodiscard]] const std::string& getName() const;
00050
```

```
00055     [[nodiscard]] const std::string& getAddress() const;
00056
00061     [[nodiscard]] const std::string& getPhoneNumber() const;
00062
00067     [[nodiscard]] const std::string& getEmail() const;
00068
00073     void updateName(std::string newName);
00074
00079     void updateAddress(std::string newAddress);
00080
00085     void updatePhoneNumber(std::string newPhoneNumber);
00086
00091     void updateEmail(std::string newEmail);
00092
00096     void display() const;
00097
00104     friend std::ostream& operator<<(std::ostream& os, const CompanyBusinessCard& BC);
00105
00112     friend std::istream& operator>>(std::istream& is, CompanyBusinessCard& BC);
00113 };
```

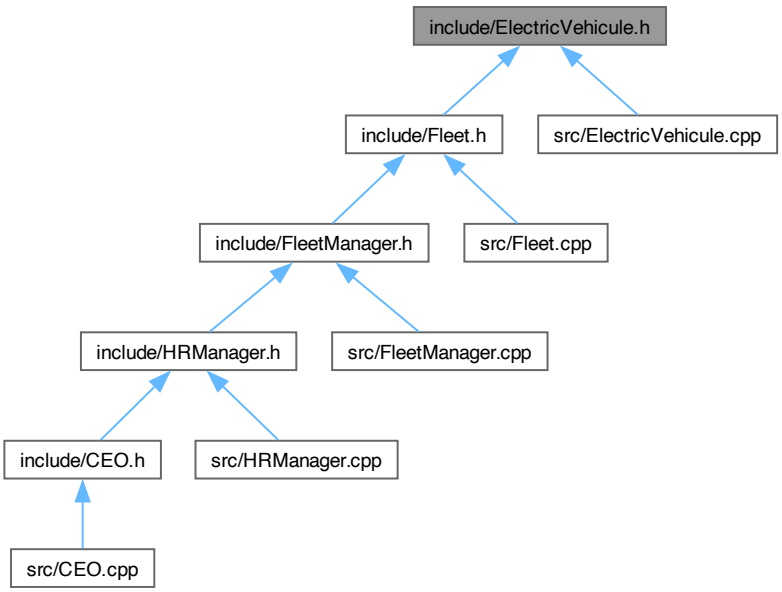
## 5.7 include/ElectricVehicule.h File Reference

```
#include <iostream>
#include "ElectricVehiculeSpecs.h"
```

Include dependency graph for ElectricVehicule.h:



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




### Classes

- class ElectricVehicule

  *Représente un véhicule électrique, dérivé de la classe Vehicule.*

## 5.8 ElectricVehicule.h

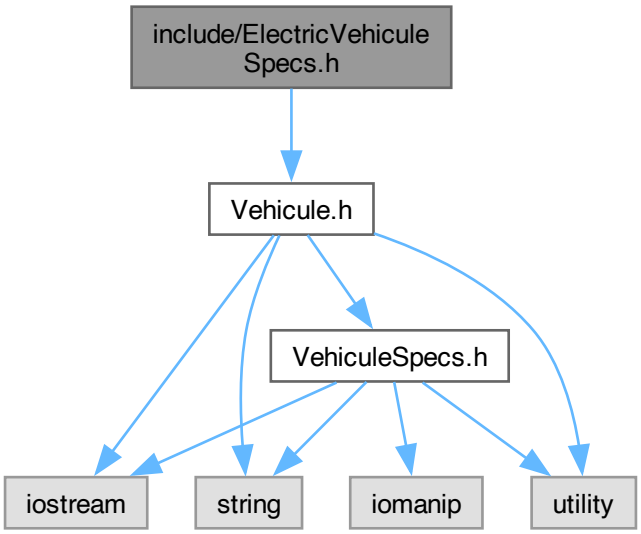
Go to the documentation of this file.

```
00001 #pragma once
00002 #include <iostream>
00003
00004 #include "ElectricVehiculeSpecs.h"
00005
00013 class ElectricVehicule : public Vehicule
00014 {
00015 private:
00016     ElectricVehiculeSpecs specs;
00018 protected:
00023     void print(std::ostream& os) const override;
00024
00025 public:
00030     explicit ElectricVehicule(ElectricVehiculeSpecs specs)
00031         : Vehicule(specs), specs(std::move(specs)) {}
00032
00037     [[nodiscard]] const ElectricVehiculeSpecs& getSpecs() override;
00038
00045     void display() const override;
00046
00053     friend std::ostream& operator«(std::ostream& os, const ElectricVehicule& eVehicule);
00054 };
```

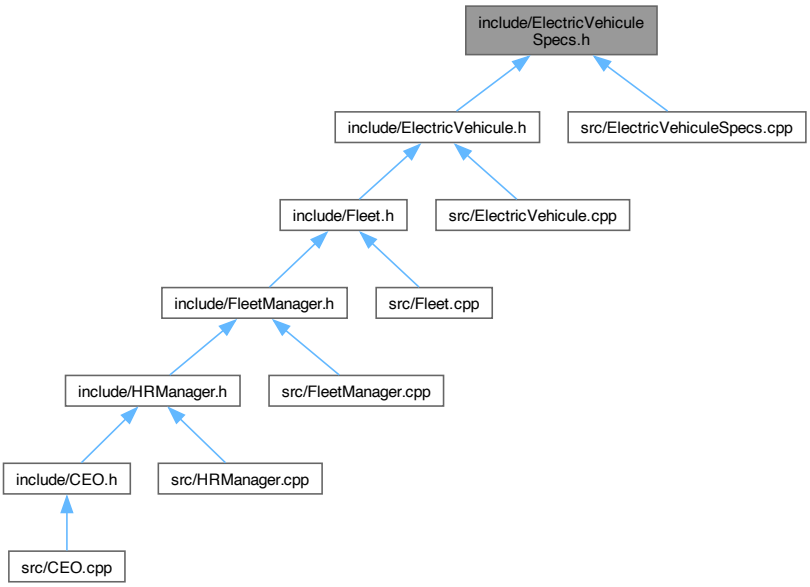
## 5.9 include/ElectricVehiculeSpecs.h File Reference

```
#include "Vehicule.h"
```

Include dependency graph for ElectricVehiculeSpecs.h:




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



### Classes

- struct ElectricVehiculeSpecs

  *Représente les spécifications spécifiques d'un véhicule électrique.*

## 5.10 ElectricVehiculeSpecs.h

Go to the documentation of this file.

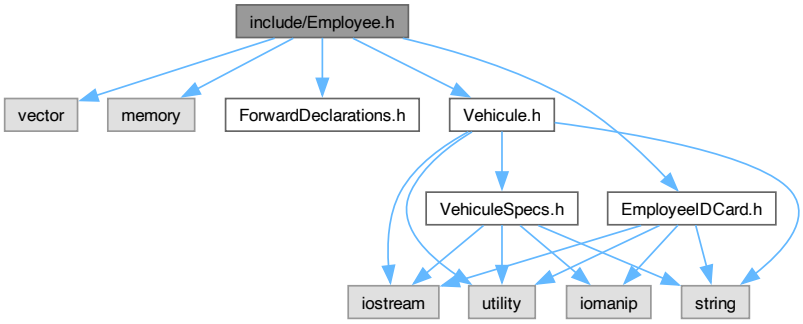
```
00001 #pragma once
00002
00003 #include "Vehicule.h"
00004
00013 struct ElectricVehiculeSpecs : VehiculeSpecs
00014 {
00015     friend class FleetManager;
00017 private:
00018     int autonomy{0};
00020 protected:
00025     void print(std::ostream& os) const override;
00026
00027 public:
00034     ElectricVehiculeSpecs(
00035         std::string brand,
00036         std::string model,
00037         int autonomy
00038         )
00039         : VehiculeSpecs(
00040             std::move(brand), std::move(model)
00041             ),
00042             autonomy(autonomy)
00043         {}
00044
00053     ElectricVehiculeSpecs(
00054         std::string brand,
00055         std::string model,
00056         unsigned int mileAge,
00057         bool booked,
00058         int autonomy
00059         ) : VehiculeSpecs(
00060                 std::move(brand),
00061                 std::move(model),
00062                 mileAge,
00063                 booked
00064             ),
00065             autonomy(autonomy)
00066         {}
00067
00072     [[nodiscard]] int getAutonomy() const;
00073
00080     void display() const override;
00081
00088     friend std::ostream& operator<<(std::ostream& os, const ElectricVehiculeSpecs& specs);
00089 };
```

## 5.11 include/Employee.h File Reference

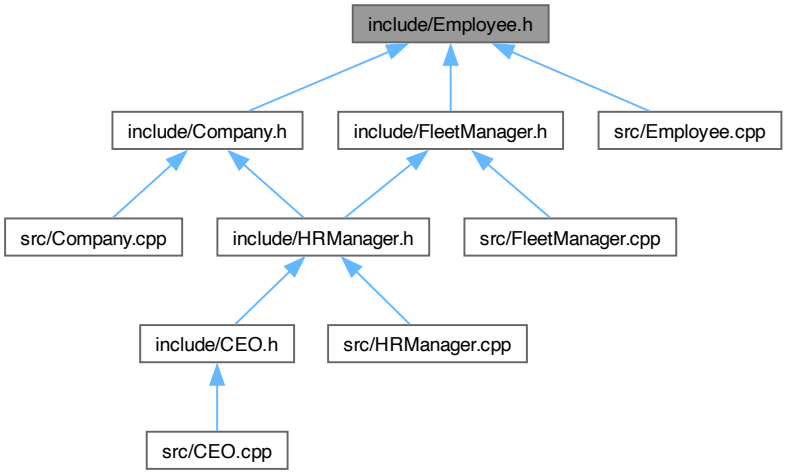
```
#include <vector>
#include <memory>
#include "ForwardDeclarations.h"
#include "EmployeeIDCard.h"
```

```
#include "Vehicule.h"
```

Include dependency graph for Employee.h:




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




### Classes

- class Employee

  *Représente un employé dans le système de gestion de flotte.*

## 5.12 Employee.h

Go to the documentation of this file.

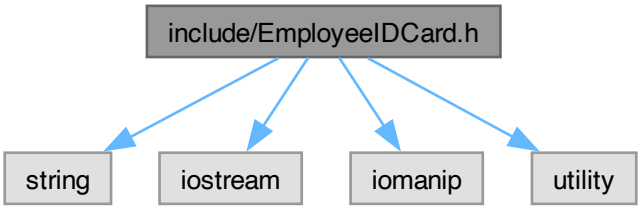
```
00001 #pragma once
00002 #include <vector>
00003 #include <memory>
00004 #include "ForwardDeclarations.h"
```

```
00005 #include "EmployeeIDCard.h"
00006 #include "Vehicule.h"
00007
00015 class Employee
00016 {
00017 private:
00018     friend class FleetManager;
00020     EmployeeIDCard IDCard;
00021     std::vector<std::shared_ptr<Vehicule» bookedVehicules;
00023 public:
00028     explicit Employee(EmployeeIDCard IDCard) : IDCard(std::move(IDCard))
00029     {
00030     }
00031
00036     [[nodiscard]] EmployeeIDCard& getIDCARD();
00037
00042     [[nodiscard]] const std::vector<std::shared_ptr<Vehicule»& getBookedVehicules() const;
00043
00049     bool hasBookedVehicule(const std::shared_ptr<Vehicule>& vehicule) const;
00050
00054     void displayBookings() const;
00055 };
```

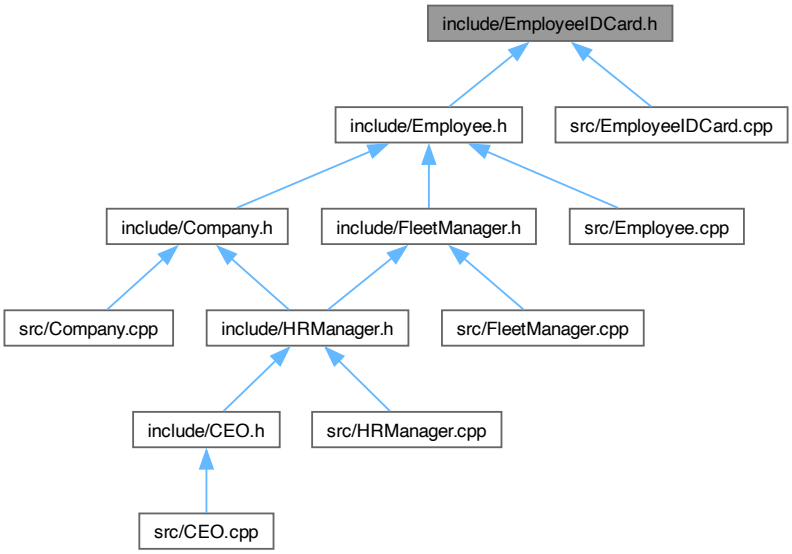
## 5.13 include/EmployeeIDCard.h File Reference

```
#include <string>
#include <iostream>
#include <iomanip>
#include <utility>
```

Include dependency graph for EmployeeIDCard.h:



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




### Classes

- struct EmployeeIDCard

  *Représente une carte d'identité pour un employé.*

## 5.14 EmployeeIDCard.h

Go to the documentation of this file.

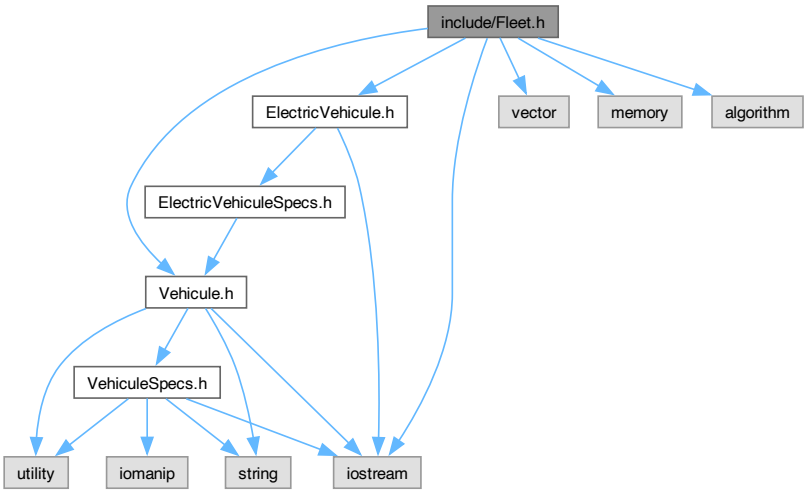
```
00001 #pragma once
00002 #include <string>
00003 #include <iostream>
00004 #include <iomanip>
00005 #include <utility>
00006
00015 struct EmployeeIDCard
00016 {
00017 private:
00018     friend class HRManager;
00020     std::string companyName;
00021     unsigned int ID;
00022     std::string fullName;
00023     std::string position;
00025 public:
00027     EmployeeIDCard() = default;
00028
00036     EmployeeIDCard(
00037         std::string companyName,
00038         unsigned int ID,
00039         std::string fullName,
00040         std::string position
00041     ) : companyName(std::move(companyName)),
00042         ID(ID),
00043         fullName(std::move(fullName)),
00044         position(std::move(position))
00045     {
00046     }
00047
00052     [[nodiscard]] double getID() const;
00053
```

```
00058     [[nodiscard]] const std::string& getFullName() const;
00059
00064     [[nodiscard]] const std::string& getPosition() const;
00065
00070     [[nodiscard]] const std::string& getCompanyName() const;
00071
00075     void display() const;
00076
00083     friend std::ostream& operator<<(std::ostream& os, const EmployeeIDCard& IDCard);
00084
00091     friend std::istream& operator>>(std::istream& is, EmployeeIDCard& IDCard);
00092 };
```

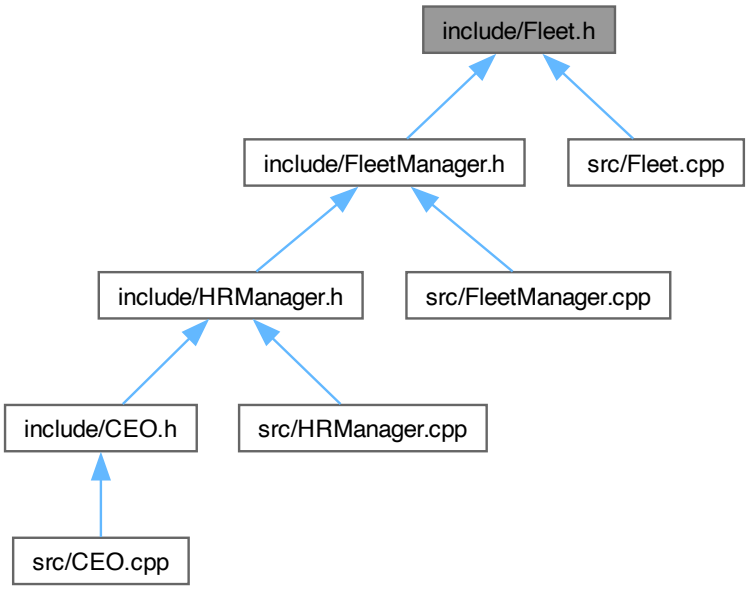
## 5.15 include/Fleet.h File Reference

```
#include <iostream>
#include <vector>
#include <memory>
#include <algorithm>
#include "Vehicule.h"
#include "ElectricVehicule.h"
```

Include dependency graph for Fleet.h:



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




### Classes

- class Fleet

  *Représente une flotte de véhicules.*

## 5.16 Fleet.h

Go to the documentation of this file.

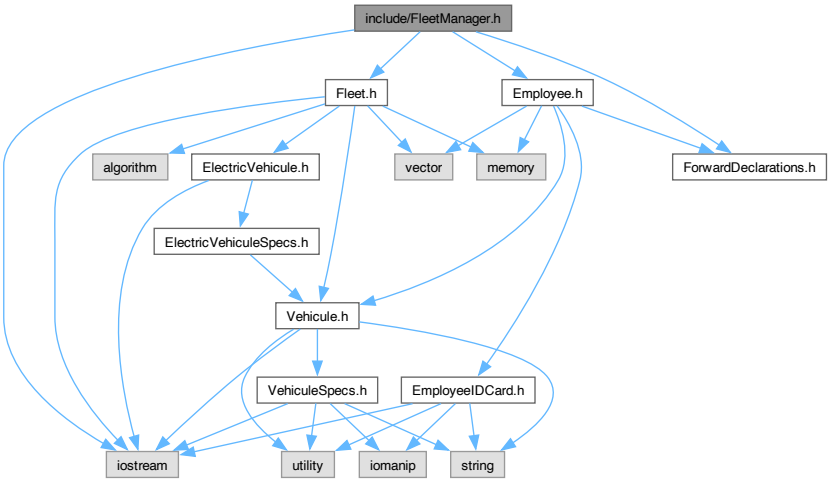
```
00001 #pragma once
00002 #include <iostream>
00003 #include <vector>
00004 #include <memory>
00005 #include <algorithm>
00006
00007 #include "Vehicule.h"
00008 #include "ElectricVehicule.h"
00009
00018 class Fleet
00019 {
00020 private:
00021     friend class FleetManager;
00023     std::vector<std::shared_ptr<Vehicule» vehicules;
00025 public:
00027     Fleet() = default;
00028
00033     explicit Fleet(std::vector<std::shared_ptr<Vehicule»&& vehicules)
00034         : vehicules(std::move(vehicules)) {}
00035
00040     [[nodiscard]] std::shared_ptr<std::vector<std::shared_ptr<Vehicule»> getVehicules() const;
00041
00047     [[nodiscard]] std::vector<std::shared_ptr<Vehicule» getBrand(const std::string& brand) const;
00048
00055     [[nodiscard]] std::shared_ptr<Vehicule> getVehicule(
00056         const std::string& brand,
00057         const std::string& model
```

```
00058     ) const;
00059
00067     [[nodiscard]] std::shared_ptr<Vehicule> getVehicule(
00068         const std::string& brand,
00069         const std::string& model,
00070         bool booked
00071     ) const;
00072
00081     [[nodiscard]] std::shared_ptr<Vehicule> getVehicule(
00082         const std::string& brand,
00083         const std::string& model,
00084         unsigned int mileAge,
00085         bool booked = false
00086     ) const;
00087
00093     [[nodiscard]] bool hasVehicule(const std::shared_ptr<Vehicule>& vehicule) const;
00094
00101     [[nodiscard]] bool hasVehicule(const std::string& brand, const std::string& model) const;
00102
00107     [[nodiscard]] std::vector<std::shared_ptr<Vehicule» getAvailableVehicules() const;
00108
00113     [[nodiscard]] std::vector<std::shared_ptr<Vehicule» getBookedVehicules() const;
00114
00118     void displayAvailableVehicules() const;
00119
00123     void displayBookedVehicules() const;
00124
00128     void display() const;
00129 };
```

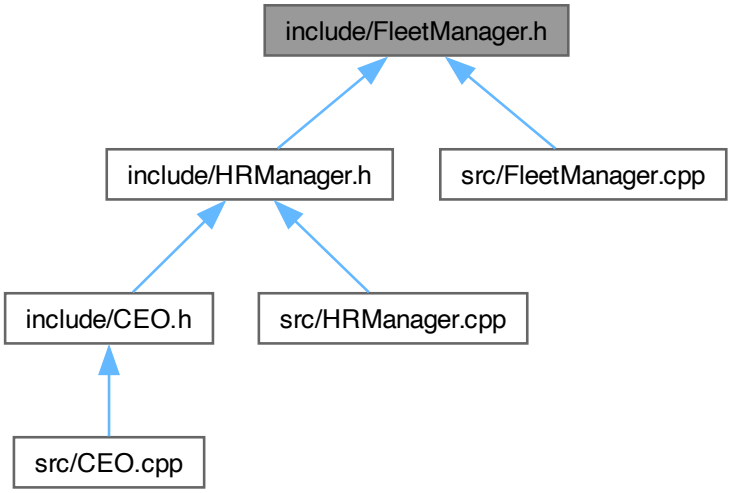
## 5.17 include/FleetManager.h File Reference

```
#include <iostream>
#include "ForwardDeclarations.h"
#include "Fleet.h"
#include "Employee.h"
```

Include dependency graph for FleetManager.h:



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




### Classes

- class FleetManager

  *Gère les véhicules de la flotte et leur assignation aux employés.*

## 5.18 FleetManager.h

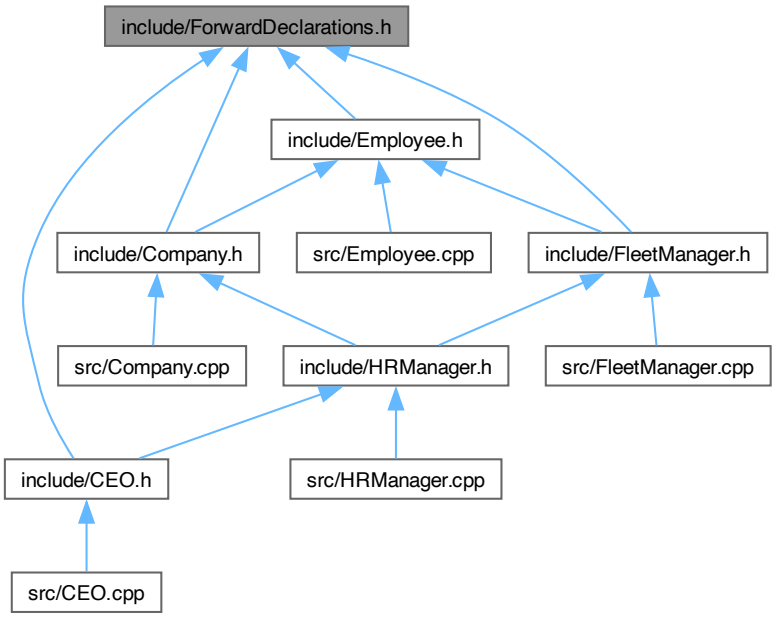
Go to the documentation of this file.

```
00001 #pragma once
00002 #include <iostream>
00003 #include "ForwardDeclarations.h"
00004 #include "Fleet.h"
00005 #include "Employee.h"
00006
00007 class HRManager;
00008
00018 class FleetManager
00019 {
00020 private:
00021     friend class HRManager;
00023     EmployeeIDCard IDCard;
00024     std::shared_ptr<Fleet> fleet;
00026 public:
00031     explicit FleetManager(
00032         EmployeeIDCard IDCard
00033     ) : IDCard(std::move(IDCard))
00034     {}
00035
00040     void addVehicule(Vehicule& vehicule);
00041
00046     void addVehicules(std::vector<Vehicule>&& vehicules);
00047
00052     void addVehicules(std::vector<ElectricVehicule>&& vehicules);
00053
00059     void assignVehiculeTo(
00060         std::shared_ptr<Vehicule> vehicule,
00061         const std::shared_ptr<Employee>& employee
00062     );
00063
```

```
00070     void assignVehiculeTo(
00071         const std::string& brand,
00072         const std::string& model,
00073         const std::shared_ptr<Employee>& employee
00074     );
00075
00081     void assignVehiculesTo(
00082         const std::vector<std::array<std::string, 2>& brandModelCouples,
00083         const std::shared_ptr<Employee>& employee
00084     );
00085
00091     void assignVehiculesTo(
00092         const std::vector<std::shared_ptr<Vehicule>& vehicules,
00093         const std::shared_ptr<Employee>& employee
00094     );
00095
00102     std::shared_ptr<Vehicule> unassignVehiculeOf(
00103         const std::shared_ptr<Vehicule>& vehicule,
00104         const std::shared_ptr<Employee>& employee
00105     );
00106
00114     std::shared_ptr<Vehicule> unassignVehiculeOf(
00115         const std::string& brand,
00116         const std::string& model,
00117         const std::shared_ptr<Employee>& employee
00118     );
00119
00126     std::vector<std::shared_ptr<Vehicule>> unassignVehiculesOf(
00127         const std::vector<std::array<std::string, 2>& brandModelCouples,
00128         const std::shared_ptr<Employee>& employee
00129     );
00130
00137     std::vector<std::shared_ptr<Vehicule>> unassignVehiculesOf(
00138         const std::vector<std::shared_ptr<Vehicule>& vehicules,
00139         const std::shared_ptr<Employee>& employee
00140     );
00141
00147     std::vector<std::shared_ptr<Vehicule>> unassignAllVehiculesOf(
00148         const std::shared_ptr<Employee>& employee
00149     );
00150
00156     void updateVehiculeMileAge(const std::shared_ptr<Vehicule>& vehicule, unsigned int newMileAge);
00157 };
```

## 5.19 include/ForwardDeclarations.h File Reference

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




## 5.20 ForwardDeclarations.h

Go to the documentation of this file.

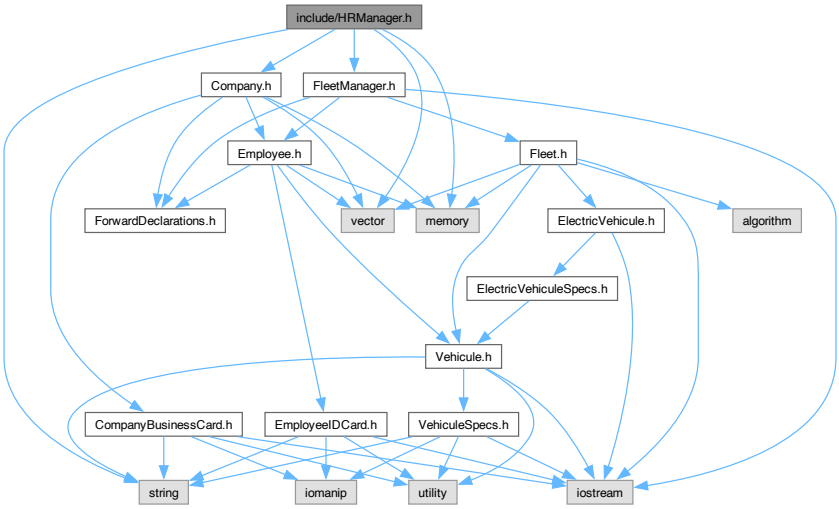
```
00001 #pragma once
00002
00003 // Declare all classes/structs here with minimal dependencies
00004 class Employee;
00005 class Company;
00006 class CEO;
00007 class HRManager;
00008 class FleetManager;
00009 class Fleet;
00010 class Vehicule;
00011 class ElectricVehicule;
00012 struct CompanyBusinessCard;
00013 struct EmployeeIDCard;
00014 struct VehiculeSpecs;
00015 struct ElectricVehiculeSpecs;
```

## 5.21 include/HRManager.h File Reference

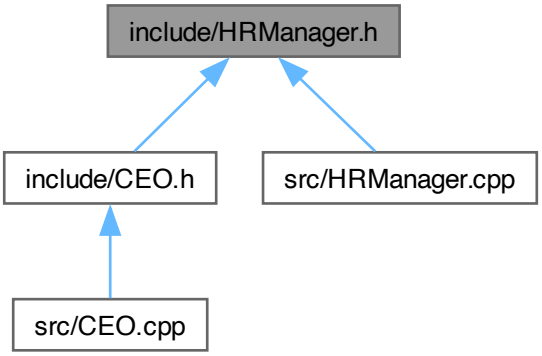
```
#include <string>
#include <memory>
#include <vector>
#include "Company.h"
```

```
#include "FleetManager.h"
```

Include dependency graph for HRManager.h:




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




### Classes

- class HRManager

  *Gère les employés d'une entreprise, y compris l'embauche, le licenciement et l'affectation du gestionnaire de flotte.*

## 5.22 HRManager.h

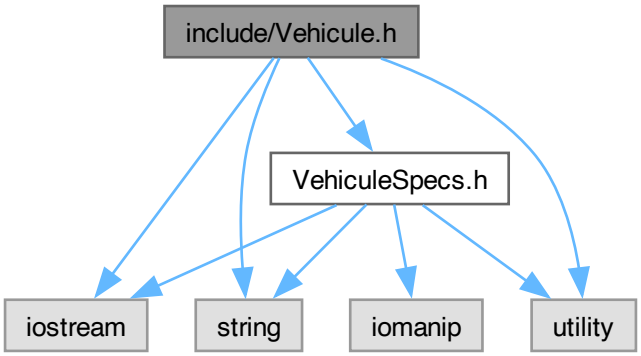
Go to the documentation of this file.

```
00001 #pragma once
00002 #include <string>
00003 #include <memory>
00004 #include <vector>
00005
00006 #include "Company.h"
00007 #include "FleetManager.h"
00008
00009 class Company;
00010
00011 class CEO;
00012
00020 class HRManager
00021 {
00022 private:
00023     friend class CEO;
00025     std::shared_ptr<Company> company;
00026     EmployeeIDCard IDCard;
00028 public:
00033     explicit HRManager(EmployeeIDCard IDCard) : IDCard(std::move(IDCard))
00034     {}
00035
00040     [[nodiscard]] const EmployeeIDCard& getIDCard() const;
00041
00046     void setFleetManager(FleetManager&& fleetManager);
00047
00052     void employEmployee(Employee&& employee);
00053
00058     void employEmployees(std::vector<Employee>&& employees);
00059
00065     std::shared_ptr<Employee> fireEmployee(
00066         const std::shared_ptr<Employee>& employee
00067     );
00068
00074     std::vector<std::shared_ptr<Employee» fireEmployees(
00075         std::vector<std::shared_ptr<Employee»&& employees
00076     );
00077 };
```

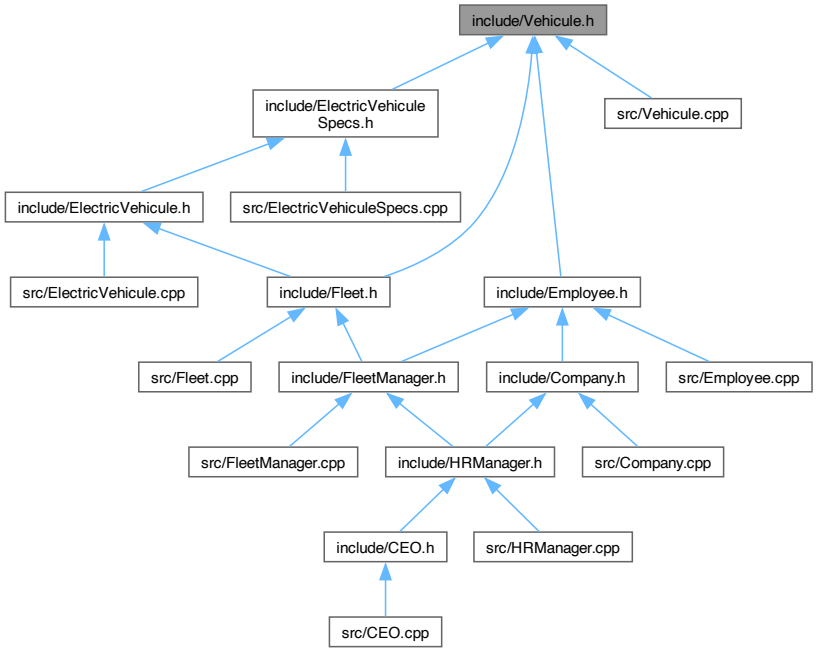
## 5.23 include/Vehicule.h File Reference

```
#include <iostream>
#include <string>
#include <utility>
#include "VehiculeSpecs.h"
```

Include dependency graph for Vehicule.h:



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




### Classes

- class Vehicule

  *Représente un véhicule avec des spécifications.*

## 5.24 Vehicule.h

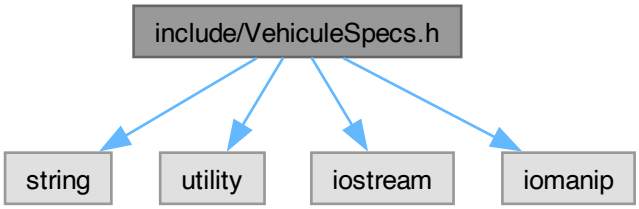
Go to the documentation of this file.

```
00001 #pragma once
00002 #include <iostream>
00003 #include <string>
00004 #include <utility>
00005
00006 #include "VehiculeSpecs.h"
00007
00016 class Vehicule
00017 {
00018 private:
00019     friend class FleetManager;
00020     VehiculeSpecs specs;
00022 protected:
00027     virtual void print(std::ostream& os) const;
00028
00029 public:
00034     explicit Vehicule(VehiculeSpecs specs) : specs(std::move(specs))
00035     {
00036     }
00037
00042     [[nodiscard]] virtual const VehiculeSpecs& getSpecs();
00043
00050     virtual void display() const;
00051
00058     friend std::ostream& operator<<(std::ostream& os, const Vehicule& vehicule);
00059
00061     virtual ~Vehicule() = default;
00062 };
```

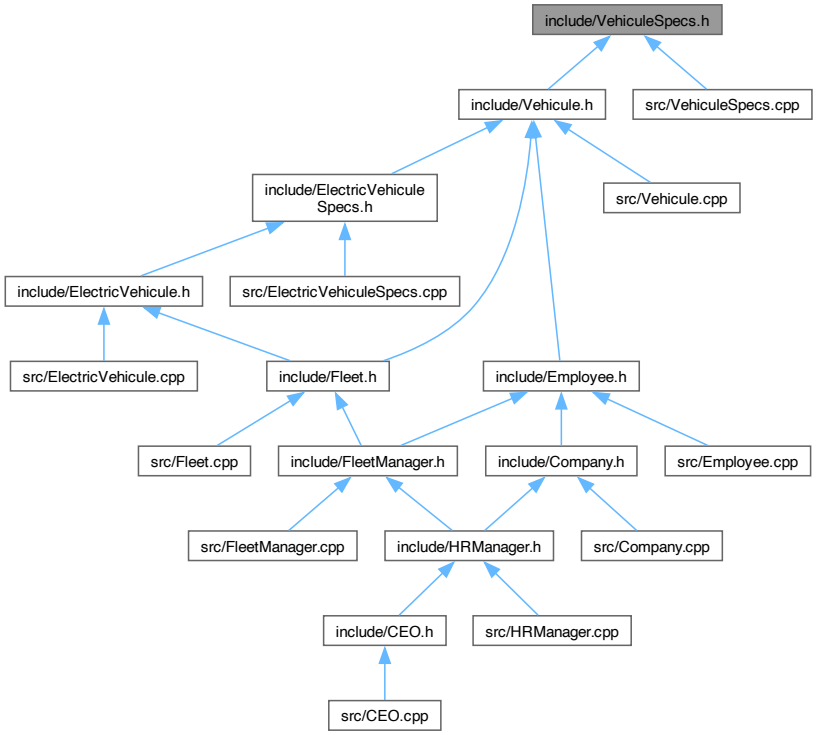
## 5.25 include/VehiculeSpecs.h File Reference

```
#include <string>
#include <utility>
#include <iostream>
#include <iomanip>
```

Include dependency graph for VehiculeSpecs.h:




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




### Classes

- struct VehiculeSpecs

  *Représente les spécifications de base d'un véhicule.*

# 5.26 VehiculeSpecs.h

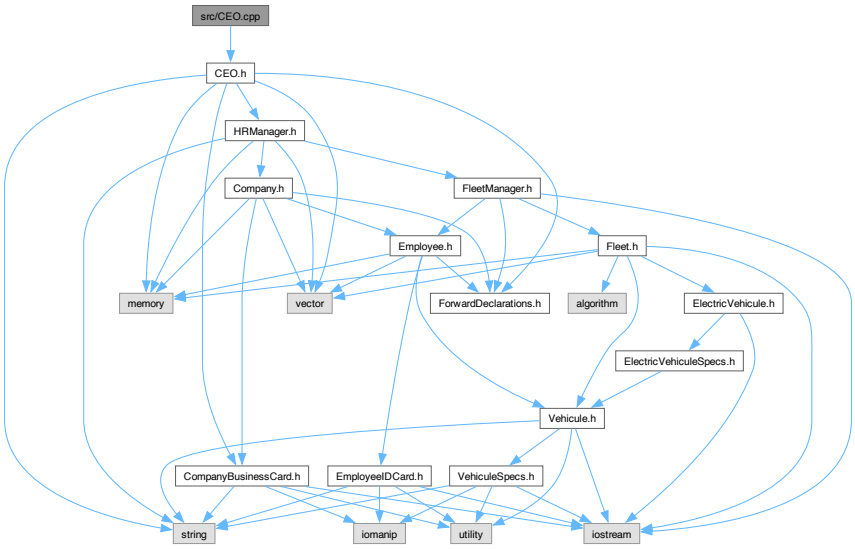
Go to the documentation of this file.

```
00001 #pragma once
00002 #include <string>
00003 #include <utility>
00004 #include <iostream>
00005 #include <iomanip>
00006
00007 class FleetManager;
00008
00017 struct VehiculeSpecs
00018 {
00019     friend class FleetManager;
00021 private:
00022     const std::string brand;
00023     const std::string model;
00024     unsigned int mileAge{};
00025     bool booked{};
00027 protected:
00032     virtual void print(std::ostream& os) const;
00033
00034 public:
00036     virtual ~VehiculeSpecs() = default;
00037
00043     VehiculeSpecs(std::string brand, std::string model)
00044         : brand(std::move(brand)), model(std::move(model))
00045     {
00046     }
00047
00055     VehiculeSpecs(
00056         std::string brand,
00057         std::string model,
00058         unsigned int mileAge,
00059         bool booked
00060     ) : brand(std::move(brand)),
00061         model(std::move(model)),
00062         mileAge(mileAge),
00063         booked(booked)
00064     {
00065     }
00066
00071     [[nodiscard]] const std::string& getBrand() const;
00072
00077     [[nodiscard]] const std::string& getModel() const;
00078
00083     [[nodiscard]] unsigned int getMileAge() const;
00084
00089     [[nodiscard]] bool isBooked() const;
00090
00097     virtual void display() const;
00098
00105     friend std::ostream& operator<<(std::ostream& os, const VehiculeSpecs& specs);
00106 };
```

## 5.27 src/CEO.cpp File Reference

```
#include "CEO.h"
```

Include dependency graph for CEO.cpp:




### Functions

- std::istream & operator>> (std::istream &is, CEO &ceo)

  *Surcharge de l'opérateur >> pour saisir les informations du PDG.*

### 5.27.1 Function Documentation

#### 5.27.1.1 operator>>()

```
std::istream & operator>> (
            std::istream & is,
            CEO & ceo)
```

Surcharge de l'opérateur >> pour saisir les informations du PDG.

Surcharge de l'opérateur >> pour saisir les informations d'un PDG.

**Parameters**

| | |
|---|---|
| *is* | Le flux d'entrée pour saisir les informations. |
| *ceo* | L'objet PDG à remplir. |

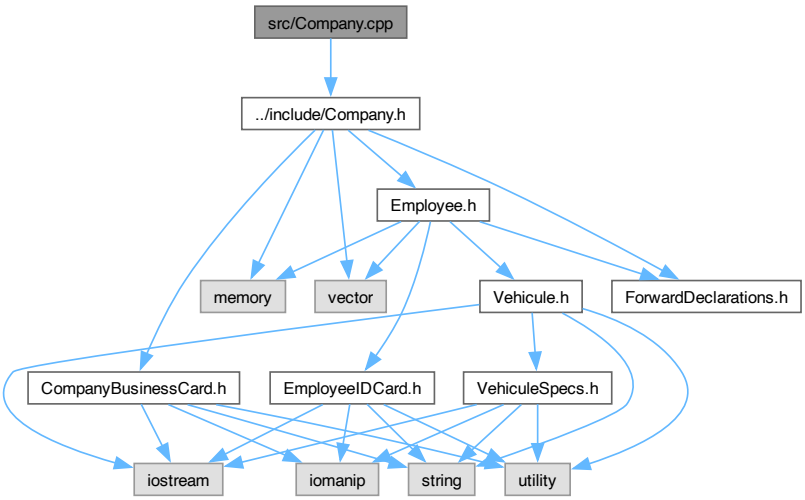
**Returns**

Une référence vers le flux d'entrée.

Le flux d'entrée est utilisé pour remplir l'objet `ceo`. Aucune copie n'est nécessaire, car on travaille directement avec la référence de l'objet `ceo`.

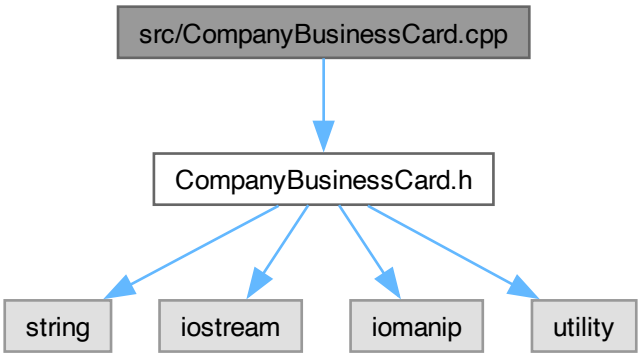
## 5.28 src/Company.cpp File Reference

`#include "../include/Company.h"`
Include dependency graph for Company.cpp:




## 5.29 src/CompanyBusinessCard.cpp File Reference

`#include "CompanyBusinessCard.h"`
Include dependency graph for CompanyBusinessCard.cpp:




### Functions

- std::ostream & operator<< (std::ostream &os, const CompanyBusinessCard &BC)

*Surcharge de l'opérateur << pour afficher une carte professionnelle.*

- std::istream & operator>> (std::istream &is, CompanyBusinessCard &BC)

  *Surcharge de l'opérateur >> pour saisir les informations de la carte professionnelle.*

## 5.29.1 Function Documentation

### 5.29.1.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & os,
            const CompanyBusinessCard & BC)
```

Surcharge de l'opérateur << pour afficher une carte professionnelle.

Surcharge de l'opérateur << pour afficher la carte professionnelle.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les informations seront imprimées. |
| *BC* | La carte professionnelle à afficher. |

**Returns**

Une référence vers le flux de sortie.

L'opérateur << est utilisé pour afficher une ligne formatée de la carte professionnelle. Aucune copie d'objet n'est effectuée ici, juste un accès direct aux membres de `CompanyBusinessCard`.

### 5.29.1.2 operator>>()

```
std::istream & operator>> (
            std::istream & is,
            CompanyBusinessCard & BC)
```

Surcharge de l'opérateur >> pour saisir les informations de la carte professionnelle.

Surcharge de l'opérateur >> pour saisir les informations de la carte.

**Parameters**

| | |
|---|---|
| *is* | Le flux d'entrée où les informations seront lues. |
| *BC* | La carte professionnelle à remplir. |

**Returns**

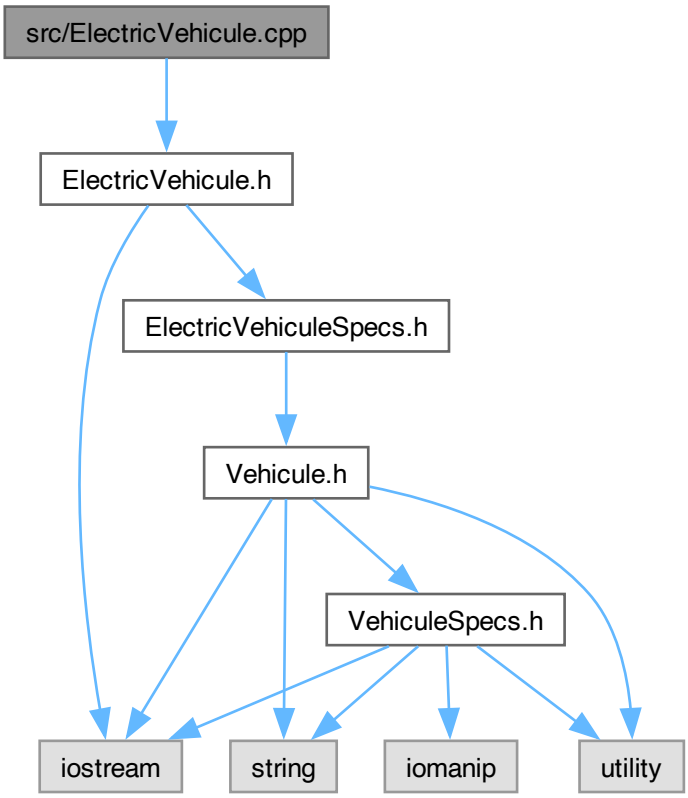
Une référence vers le flux d'entrée.

Cet opérateur permet de saisir les informations de la carte professionnelle en utilisant un flux d'entrée. Chaque champ est lu et stocké dans les attributs respectifs de `CompanyBusinessCard`.

## 5.30 src/ElectricVehicule.cpp File Reference

```
#include "ElectricVehicule.h"
```

Include dependency graph for ElectricVehicule.cpp:




### Functions

- std::ostream & operator<< (std::ostream &os, const ElectricVehicule &eVehicule)

  *Surcharge de l'opérateur << pour afficher un véhicule électrique.*

### 5.30.1 Function Documentation

#### 5.30.1.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & os,
            const ElectricVehicule & eVehicule)
```

Surcharge de l'opérateur << pour afficher un véhicule électrique.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les informations du véhicule seront imprimées. |
| *eVehicule* | Le véhicule électrique à afficher. |

**Returns**

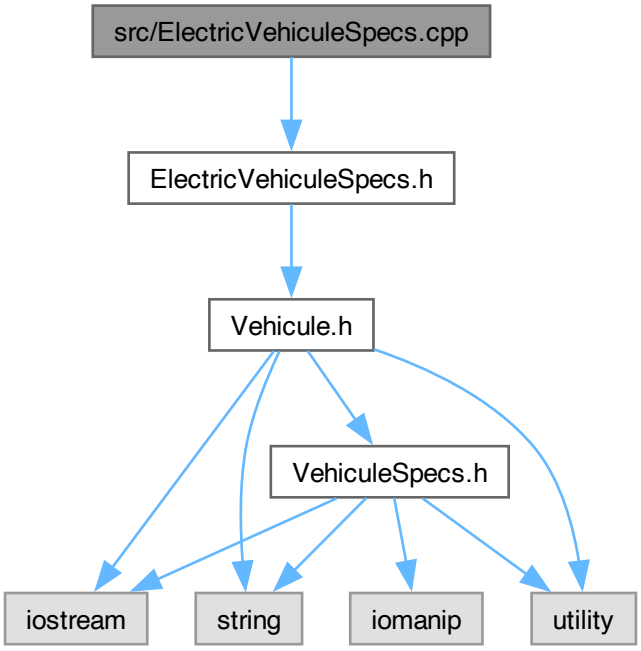
Une référence vers le flux de sortie.

Utilise `print()` pour afficher les détails du véhicule dans le flux.

## 5.31 src/ElectricVehiculeSpecs.cpp File Reference

```
#include "ElectricVehiculeSpecs.h"
```

Include dependency graph for ElectricVehiculeSpecs.cpp:




### Functions

- std::ostream & operator<< (std::ostream &os, const ElectricVehiculeSpecs &specs)

  *Surcharge de l'opérateur << pour afficher les spécifications du véhicule électrique.*

### 5.31.1 Function Documentation

#### 5.31.1.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & os,
            const ElectricVehiculeSpecs & specs)
```

Surcharge de l'opérateur << pour afficher les spécifications du véhicule électrique.

Surcharge de l'opérateur << pour afficher les spécifications d'un véhicule électrique.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les spécifications seront imprimées. |
| *specs* | Les spécifications du véhicule à afficher. |

**Returns**

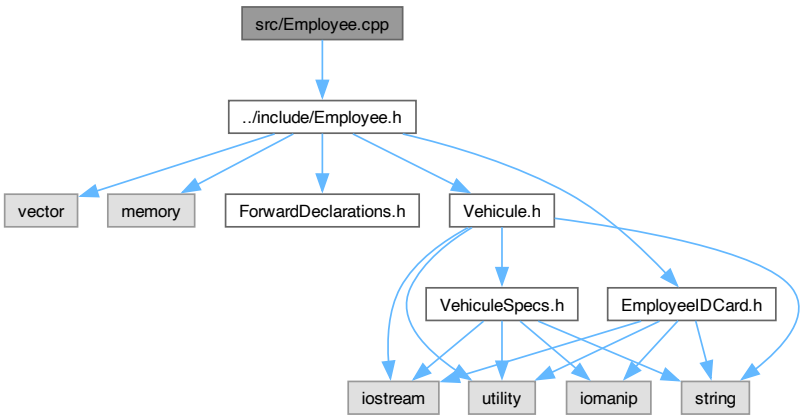
Une référence vers le flux de sortie.

L'opérateur << est utilisé pour afficher les spécifications du véhicule en appelant la méthode `print()`. Cela permet de conserver un affichage structuré.

## 5.32 src/Employee.cpp File Reference

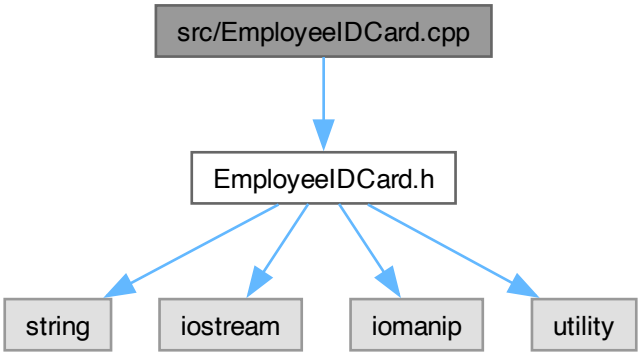
```
#include "../include/Employee.h"
```

Include dependency graph for Employee.cpp:



## 5.33 src/EmployeeIDCard.cpp File Reference

```
#include "EmployeeIDCard.h"
```
Include dependency graph for EmployeeIDCard.cpp:




### Functions

- std::ostream & operator<< (std::ostream &os, const EmployeeIDCard &IDCard)

  *Surcharge de l'opérateur << pour afficher la carte d'identité de l'employé.*
- std::istream & operator>> (std::istream &is, EmployeeIDCard &IDCard)

  *Surcharge de l'opérateur >> pour saisir les informations de la carte d'identité.*

### 5.33.1 Function Documentation

#### 5.33.1.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & os,
            const EmployeeIDCard & IDCard)
```

Surcharge de l'opérateur << pour afficher la carte d'identité de l'employé.

Surcharge de l'opérateur << pour afficher les informations de la carte d'identité.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les informations de l'employé seront imprimées. |
| *IDCard* | La carte d'identité de l'employé à afficher. |

**Returns**

Une référence vers le flux de sortie.

Cette surcharge permet d'afficher les détails de la carte d'identité dans un format structuré et aligné.

#### 5.33.1.2 operator>>()

```
std::istream & operator>> (
            std::istream & is,
            EmployeeIDCard & IDCard)
```

Surcharge de l'opérateur >> pour saisir les informations de la carte d'identité.

Surcharge de l'opérateur >> pour lire les informations de la carte d'identité.

**Parameters**

| | |
|---|---|
| *is* | Le flux d'entrée où les informations seront lues. |
| *IDCard* | La carte d'identité de l'employé à remplir. |

**Returns**

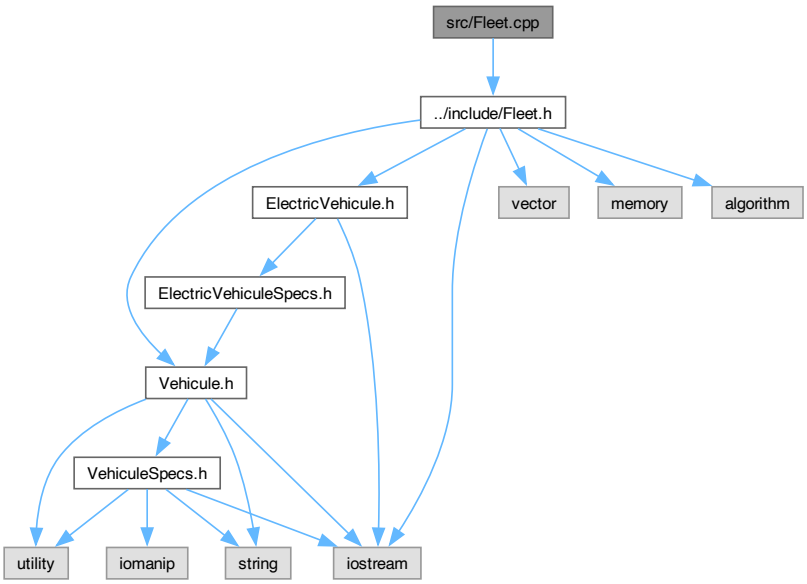
Une référence vers le flux d'entrée.

Cette surcharge permet de saisir les informations de la carte d'identité de l'employé via le flux d'entrée, en demandant les différentes informations comme le nom, l'ID, etc.

## 5.34 src/Fleet.cpp File Reference

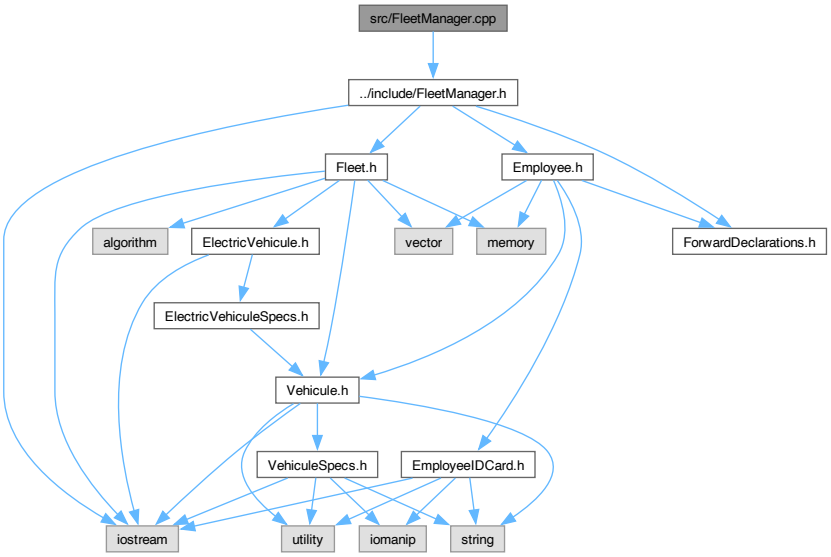
```
#include "../include/Fleet.h"
```

Include dependency graph for Fleet.cpp:



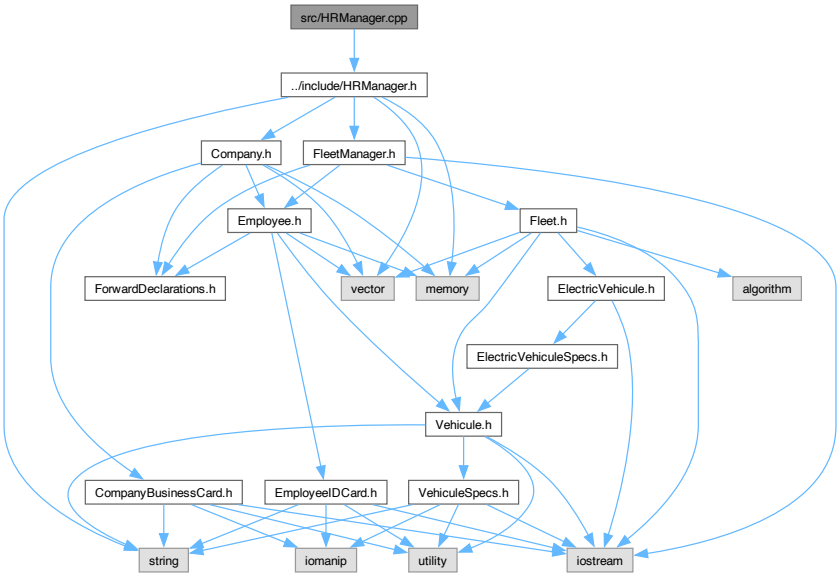
## 5.35 src/FleetManager.cpp File Reference

`#include "../include/FleetManager.h"`
Include dependency graph for FleetManager.cpp:



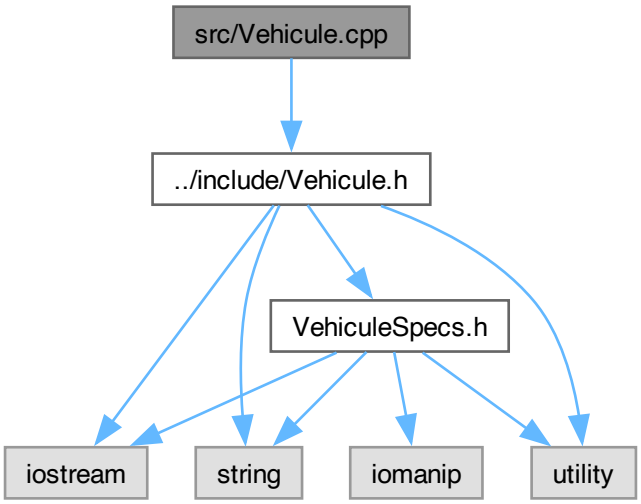

## 5.36 src/HRManager.cpp File Reference

`#include "../include/HRManager.h"`
Include dependency graph for HRManager.cpp:



## 5.37 src/Vehicule.cpp File Reference

`#include "../include/Vehicule.h"`
Include dependency graph for Vehicule.cpp:




### Functions

- std::ostream & operator<< (std::ostream &os, const Vehicule &vehicule)
  *Surcharge de l'opérateur << pour afficher un véhicule.*

### 5.37.1 Function Documentation

#### 5.37.1.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & os,
            const Vehicule & vehicule)
```

Surcharge de l'opérateur << pour afficher un véhicule.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les informations du véhicule seront imprimées. |
| *vehicule* | Le véhicule à afficher. |

**Returns**

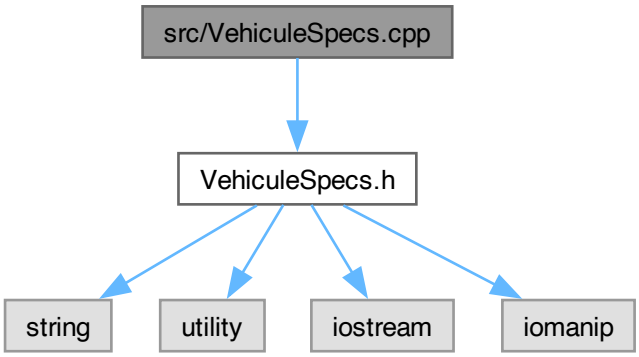
Une référence vers le flux de sortie.

L'opérateur << est utilisé pour afficher un véhicule en appelant la méthode `print()`. Cela permet un affichage structuré des informations du véhicule sans effectuer de copie.

## 5.38 src/VehiculeSpecs.cpp File Reference

```
#include "VehiculeSpecs.h"
```

Include dependency graph for VehiculeSpecs.cpp:




### Functions

- std::ostream & operator<< (std::ostream &os, const VehiculeSpecs &specs)

  *Surcharge de l'opérateur << pour afficher les spécifications du véhicule.*

### 5.38.1 Function Documentation

#### 5.38.1.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & os,
            const VehiculeSpecs & specs)
```

Surcharge de l'opérateur << pour afficher les spécifications du véhicule.

Surcharge de l'opérateur << pour afficher les spécifications d'un véhicule.

**Parameters**

| | |
|---|---|
| *os* | Le flux de sortie où les spécifications seront imprimées. |
| *specs* | Les spécifications du véhicule à afficher. |

**Returns**

Une référence vers le flux de sortie.

L'opérateur << est utilisé pour afficher les spécifications du véhicule de manière structurée, en appelant la méthode `print()`.