Person

1.0

Généré par Doxygen 1.8.13

Table des matières

1 Une classe pour représenter une personne

Un bon point d'entrée est celui de la documentation de l'espace de nom nvs.

2 Index des espaces de nommage

2.1 Liste des espaces de nommage

Liste de tous les espaces de nommage documentés avec une brève description :

3 Index des classes

3.1 Liste des classes

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

4 Index des fichiers

4.1 Liste des fichiers

Liste de tous les fichiers documentés avec une brève description :

5 Documentation des espaces de nommage

5.1 Référence de l'espace de nommage nvs

Espace de nom de Nicolas Vansteenkiste.

Classes

- class Date
  Classe représentant une date.
- class Person
  Classe représentant de manière extrêmement simplifiée une personne.

Énumérations

- enum Sex {
  Sex : :UNKNOWN, Sex : :FEMALE, Sex : :MALE, Sex : :VOID,
  Sex : :OTHER }
  Énumération fortement typée pour représenter le sexe d'une personne.
- enum FormOption { FormOption : :NONE = 0b0000, FormOption : :SHORT = 0b0001, FormOption : :LONG = 0b0010 }
  Énumération pour spécifier les options de mise en forme d'impression ou de conversion binaire → chaîne de caractères.

Fonctions

- **std : :ostream** & operator<< (**std : :ostream** &out, const Person &in)
  Opérateur d'injection d'une Person dans un flux en sortie.
- **std : :string** to_string (const Person &person, nvs : :FormOption fo=nvs : :FormOption : :SHORT)
  *Fonction de conversion d'une Person en **std : :string**.*
- **std : :ostream** & operator<< (**std : :ostream** &out, Sex in)
  Opérateur d'injection d'une variable ou valeur nvs : :Sex dans un flux en sortie.
- **std : :string** to_string (Sex sex, FormOption fo=FormOption : :SHORT)
  *Fonction de conversion d'une variable / valeur nvs : :Sex en **std : :string**.*
- **std : :ostream** & operator<< (**std : :ostream** &out, const Date &in)
  Opérateur d'injection d'une Date dans un flux en sortie.
- bool operator< (const Date &lhs, const Date &rhs)
  Opérateur de comparaison de deux dates.
- bool operator== (const Date &lhs, const Date &rhs)
  Opérateur de test d'égalité de deux dates.
- **std : :string** to_string (const Date &in)
  *Fonction de conversion d'une nvs : :Date en **std : :string**.*
- Date today ()
  Fonction retournant la date actuelle (au moment de l'exécution) sous la forme d'une Date.
- bool leapYear (int year)
  Fonction pour déterminer si une année est bissextile.
- template<typename T >
  bool between (const T &value, T min, T max)
  Évaluation si une valeur est comprise entre deux autres.
- template<typename T >
  std : :string error_message (const T &value, T min, T max, const **std : :string** &msg="error : ")
  Production d'un message d'erreur.
- template<typename T >
  T validate (const T &value, const T &min, const T &max, const **std : :string** &msg="error : ")
  Validation d'une valeur entre deux bornes.

5.1.1 Description détaillée

Espace de nom de Nicolas Vansteenkiste.

5.1.2 Documentation du type de l'énumération

5.1.2.1 FormOption

```
enum nvs::FormOption  [strong]
```

Énumération pour spécifier les options de mise en forme d'impression ou de conversion binaire → chaîne de caractères.

Valeurs énumérées

NONE	Absence de spécification.
SHORT	Forme courte.
LONG	Forme longue.

5.1.2.2 Sex

```
enum nvs::Sex  [strong]
```

Énumération fortement typée pour représenter le sexe d'une personne.

Valeurs énumérées

UNKNOWN	Indique que la valeur du sexe est inconnue.
FEMALE	Indique que la valeur du sexe est celle d'une femme.
MALE	Indique que la valeur du sexe est celle d'un homme.
VOID	Indique l'absence de valeur pour le sexe.
OTHER	Indique que la valeur du sexe n'est ni Sex : :UNKNOWN, ni Sex : :FEMALE, ni Sex : :MALE, ni Sex : :VOID.

5.1.3 Documentation des fonctions

5.1.3.1 between()

```
template<typename T >
bool nvs::between (
            const T & value,
            T min,
            T max )
```

Évaluation si une valeur est comprise entre deux autres.

Les opérateurs < et == doivent être disponibles pour le type `T`.

Paramètres

value	valeur à tester.
min	borne minimale.
max	borne maximale.

Renvoie

`true` si `min <= value <= max`, `false` sinon.

5.1.3.2 error_message()

```
template<typename T >
std::string nvs::error_message (
            const T & value,
            T min,
            T max,
            const  std::string & msg = "error : " )
```

Production d'un message d'erreur.

La fonction `to_string()` doit être disponible pour le type `T`.

Paramètres

value	valeur.
min	borne minimale.
max	borne maximale.
msg	en-tête du message.

Renvoie

message d'erreur.

Voir également

validate()

5.1.3.3 leapYear()

```
bool nvs::leapYear (
            int year )  [inline]
```

Fonction pour déterminer si une année est bissextile.

Paramètres

year	l'année à tester.

Renvoie

`true` si `year` est bissextile, `false` sinon.

Voir également

Date : :leapYear().

5.1.3.4 operator<()

```
bool nvs::operator< (
            const Date & lhs,
            const Date & rhs )  [inline]
```

Opérateur de comparaison de deux dates.

La valeur `true` est retournée si `lhs` est *strictement antérieure* à `rhs`. Lorsque `lhs` est *égale ou postérieure* à `rhs`, `false` est retourné.

Paramètres

lhs	une Date.
rhs	une autre Date.

Renvoie

`true` si lhs est strictement antérieure à rhs, `false` sinon.

5.1.3.5 operator<<() [1/3]

```
std::ostream & nvs::operator<< (
            std::ostream & out,
          Sex in )  [inline]
```

Opérateur d'injection d'une variable ou valeur nvs : :Sex dans un flux en sortie.

La valeur injectée correspond à la *forme courte* retournée par la fonction to_string(Sex, FormOption).

Paramètres

out	le flux dans lequel réaliser l'injection.
in	la variable ou la valeur à injecter.

Renvoie

le flux après injection.

5.1.3.6 operator<<() [2/3]

```
std::ostream& nvs::operator<< (
            std::ostream & out,
          const Date & in )
```

Opérateur d'injection d'une Date dans un flux en sortie.

La Date est injectée sous la forme `jj/mm/aaaa`, par exemple `26/04/2016`.

Il s'agit de la même forme que celle retournée par nvs : :Date : :to_string().

Paramètres

out	le flux dans lequel la Date est injectée.
in	la Date injectée dans le flux en sortie.

Renvoie

le flux dans lequel la Date a été injectée.

5.1.3.7 operator<<() [3/3]

```
std::ostream& nvs::operator<< (
            std::ostream & out,
          const Person & in )
```

Opérateur d'injection d'une Person dans un flux en sortie.

La valeur du sexe est injectée sous sa forme courte, comme pour l'opérateur d'injection d'une enum class Sex dans un flux en sortie.

Paramètres

out	flux en sortie.
in	Person à injecter.

Renvoie

le flux après l'injection.

Voir également

nvs : :operator<<(std : :ostream &, const nvs : :Date &).
nvs : :operator<<(std : :ostream &, nvs : :Sex).

5.1.3.8 operator==()

```
bool nvs::operator== (
            const Date & lhs,
            const Date & rhs )  [inline]
```

Opérateur de test d'égalité de deux dates.

Paramètres

lhs	une Date.
rhs	une autre Date.

Renvoie

`true` si les deux dates sont identiques, `false` sinon.

5.1.3.9 to_string() [1/3]

```
std::string nvs::to_string (
            Sex sex,
            FormOption fo = FormOption::SHORT )
```

Fonction de conversion d'une variable / valeur nvs : :Sex en **std : :string**.

En forme courte (`fo` *différent* de FormOption : :LONG), la **`std::string`** retournée est :
- `u` pour la valeur Sex : :UNKNOWN ;
- `f` pour la valeur Sex : :FEMALE ;
- `m` pour la valeur Sex : :MALE ;
- `v` pour la valeur Sex : :VOID ;

— o pour la valeur Sex : :OTHER.

C'est la même forme que celle injectée par l'opérateur d'injection dans un flux en sortie.

En forme longue (fo explicitement égal à FormOption : :LONG), la **std::string** retournée est :

— unknown pour la valeur Sex : :UNKNOWN ;
— female pour la valeur Sex : :FEMALE ;
— male pour la valeur Sex : :MALE ;
— void pour la valeur Sex : :VOID ;
— other pour la valeur Sex : :OTHER.

Paramètres

sex	la valeur de nvs : :Sex à convertir.
fo	pour contrôler la forme courte ou longue de la **std::string** retournée.

Renvoie

une représentation de la valeur de sex sous la forme d'une **std::string**.

5.1.3.10 to_string() [2/3]

```
std::string nvs::to_string (
            const Date & in )  [inline]
```

Fonction de conversion d'une nvs : :Date en **std : :string**.

La **std::string** retournée est "jj/mm/aaaa", par exemple "26/04/2016".

Il s'agit de la même forme que celle lors de l'injection d'une Date dans un flux en sortie (voir operator<<(std↩ : :ostream &, const Date &)).

L'appel :

```
to_string(in);
```

est équivalent à celui-ci :

```
in.to_string();
```

Paramètres

in	la Date à convertir.

Renvoie

une représentation de la date courante sous la forme d'une **std : :string**.

Voir également

Date : :to_string().

5.1.3.11 to_string() [3/3]

```
std::string nvs::to_string (
            const Person & person,
            nvs::FormOption fo = nvs::FormOption::SHORT )
```

Fonction de conversion d'une Person en **std : :string**.

Cette fonction utilise les fonctions `nvs::to_string(const Date &)` et `nvs::to_string(Sex, FormOption)` avec `fo` comme valeur pour le paramètre FormOption de cette dernière fonction.

Paramètres

person	personne à produire sous la forme d'une **std : :string**.
fo	pour contrôler la forme courte ou longue de la **std : :string** retournée.

Renvoie

une représentation de la personne sous la forme d'une **std : :string**.

Voir également

`nvs::to_string(const Date &)`.
`nvs::to_string(Sex, FormOption)`.

5.1.3.12 today()

```
Date nvs::today ( )
```

Fonction retournant la date actuelle (au moment de l'exécution) sous la forme d'une Date.

Renvoie

la date au moment de l'exécution comme une Date.

Voir également

http://en.cppreference.com/w/cpp/chrono/time_point
http://en.cppreference.com/w/cpp/chrono/c/time
http://en.cppreference.com/w/cpp/chrono/c/localtime

5.1.3.13 validate()

```
template<typename T >
T nvs::validate (
            const T & value,
            const T & min,
            const T & max,
            const  std::string & msg = "error : " )
```

Validation d'une valeur entre deux bornes.

Paramètres

value	valeur à valider.
min	borne minimale.
max	borne maximale.
msg	en-tête du message en cas de problème.

Renvoie

`value` si `min <= value <= max`.

Exceptions

std : :invalid_argument	si value $\notin$ [`min`, `max`]

Voir également

between()

6 Documentation des classes

6.1 Référence de la classe nvs : :Date

Classe représentant une date.

`#include <date.h>`

Fonctions membres publiques

- Date (int year, unsigned month, unsigned day)
  Constructeur.
- int year () const
  Accesseur en lecture de l'année.
- unsigned month () const
  Accesseur en lecture du mois.
- unsigned day () const
  Accesseur en lecture du jour.
- bool leapYear () const
  Méthode pour déterminer si l'année courante est bissextile.
- unsigned dayNumberInMonth () const
  Méthode pour connaître le nombre de jours du mois courant.
- **std : :string** to_string () const
  Méthode pour obtenir la date courante sous la forme d'une ***std : :string****.*

Attributs publics statiques

- static constexpr int MINIMUM_YEAR { **std : :numeric_limits**<int> : :min() }
  Valeur minimale acceptée pour l'année.
- static constexpr int MAXIMUM_YEAR { **std : :numeric_limits**<int> : :max() }
  Valeur maximale acceptée pour l'année.
- static constexpr unsigned MINIMUM_MONTH { 1 }
  Valeur minimale acceptée pour le mois.
- static constexpr unsigned MAXIMUM_MONTH { 12 }
  Valeur maximale acceptée pour le mois.
- static constexpr unsigned MINIMUM_DAY { 1 }
  Valeur minimale acceptée pour le jour.
- static constexpr **std : :array**< unsigned, MAXIMUM_MONTH+1 > MAXIMUM_DAY {{ 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }}
  Nombre de jours par mois, hors années bissextiles.

Attributs privés

- const int year_
  L'année.
- const unsigned month_
  Le mois.
- const unsigned day_
  Le jour.

6.1.1 Description détaillée

Classe représentant une date.

On ne se soucie pas ici du type de calendrier et de sa pertinence historique. Il n'y a pas d'années minimale ou maximale autre que celles impliquées par la taille du type sous-jacent.

Les attributs year_, month_ et day_ sont constants. Les instances de Date sont donc `immuables`.

6.1.2 Documentation des constructeurs et destructeur

6.1.2.1 Date()

```
nvs::Date::Date (
            int year,
            unsigned month,
            unsigned day )
```

Constructeur.

Ce constructeur valide ses arguments à l'aide du modèle de fonction validate(). Dès lors, une exception de type **`std::invalid_argument`** peut être levée lors de son utilisation.

On considère qu'il y a une `année 0`.

Paramètres

year	l'année.
month	le mois.
day	le jour.

Exceptions

std : :invalid_argument	si : — year $\notin$ [Date : :MINIMUM_YEAR, Date : :MAXIMUM_YEAR] — month $\notin$ [Date : :MINIMUM_MONTH, Date : :MAXIMUM_MONTH] — day $\notin$ [Date : :MINIMUM_DAY, Date : :MAXIMUM_DAY[month]] tenant compte des années bissextiles.

Voir également

year_, month_, day_.

6.1.3 Documentation des fonctions membres

6.1.3.1 day()

```
unsigned nvs::Date::day ( ) const  [inline]
```

Accesseur en lecture du jour.

Renvoie

le jour.

6.1.3.2 dayNumberInMonth()

```
unsigned nvs::Date::dayNumberInMonth ( ) const  [inline]
```

Méthode pour connaître le nombre de jours du mois courant.

Renvoie

le nombre de jours du mois courant.

6.1.3.3 leapYear()

```
bool nvs::Date::leapYear ( ) const  [inline]
```

Méthode pour déterminer si l'année courante est bissextile.

Renvoie

`true` si l'année courante est bissextile, `false` sinon.

Voir également

leapYear(int).

6.1.3.4 month()

```
unsigned nvs::Date::month ( ) const  [inline]
```

Accesseur en lecture du mois.

Renvoie

le mois.

6.1.3.5 to_string()

```
std::string nvs::Date::to_string ( ) const
```

Méthode pour obtenir la date courante sous la forme d'une **std : :string**.

La **std::string** retournée est "jj/mm/aaaa", par exemple "26/04/2016".

Il s'agit de la même forme que celle lors de l'injection d'une Date dans un flux en sortie (voir operator<<(std↩ : :ostream &, const Date &)).

Renvoie

une représentation de la date courante sous la forme d'une **std : :string**.

Voir également

to_string(const Date &).

6.1.3.6 year()

```
int nvs::Date::year ( ) const  [inline]
```

Accesseur en lecture de l'année.

Renvoie

l'année.

6.1.4 Documentation des données membres

6.1.4.1 day_

```
const unsigned nvs::Date::day_  [private]
```

Le jour.

Cet attribut est constant.

Le premier jour du mois a comme valeur MINIMUM_DAY, le suivant MINIMUM_DAY + 1, etc., jusqu'au dernier du mois, dépendant du mois et de l'année et fourni par dayNumberInMonth().

6.1.4.2 MAXIMUM_DAY

```
constexpr std::array< unsigned, MAXIMUM_MONTH + 1 > nvs::Date::MAXIMUM_DAY {{ 0, 31, 28, 31,
30, 31, 30, 31, 31, 30, 31, 30, 31 }} [static]
```

Nombre de jours par mois, *hors* années bissextiles.

Il s'agit des valeurs maximales acceptées pour le jour en fonction du mois, *sans* tenir compte des années bissextiles. Pour connaître le nombre de jours d'un mois en tenant compte des années bissextiles, il faut utiliser la méthode dayNumberInMonth().

La valeur initiale nulle est présente pour permettre d'utiliser directement la valeur du mois comme index de la **std::array**.

6.1.4.3 month_

```
const unsigned nvs::Date::month_ [private]
```

Le mois.

Cet attribut est constant.

Janvier a comme valeur MINIMUM_MONTH, février MINIMUM_MONTH + 1, etc., jusquà décembre de valeur MAXIMUM_MONTH.

6.1.4.4 year_

```
const int nvs::Date::year_ [private]
```

L'année.

Cet attribut est constant.

Elle est comprise entre MINIMUM_YEAR et MAXIMUM_YEAR, ces valeurs comprises.

On considère qu'il y a une année 0.

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

— date.h

6.2 Référence de la classe nvs : :Person

Classe représentant de manière extrêmement simplifiée une personne.

```
#include <person.h>
```

Fonctions membres publiques

- Person (const **std : :vector**< **std : :string** > &firstnames, const **std : :string** &name, Sex sex, const Date &birthdate)
  Constructeur de Person.
- Person (const **std : :vector**< **std : :string** > &firstnames, const **std : :string** &name, Sex sex, int birthdateyear, unsigned birthdatemonth, unsigned birthdateday)
  Constructeur de Person.
- Date birthdate () const
  Accesseur en lecture de la date de naissance.
- Sex sex () const
  Accesseur en lecture du sexe de la personne.
- **std : :string** name () const
  Accesseur en lecture du nom de la personne.
- **std : :vector**< **std : :string** > firstnames () const
  Accesseur en lecture des prénoms de la personne.
- unsigned firstnamesCount () const
  Accesseur en lecture du nombre de prénoms de la personne.
- **std : :string** firstname (unsigned index=0) const
  Accesseur en lecture d'un prénom en position d'index donné de la personne.
- void sex (Sex newSex)
  Accesseur en écriture du sexe de la personne.
- void name (const **std : :string** &newName)
  Accesseur en écriture du nom de la personne.
- void firstnames (const **std : :vector**< **std : :string** > &newFirstnames)
  Accesseur en écriture pour modifier la liste des prénoms de la personne.
- void addFirstname (const **std : :string** &newFirstname)
  Accesseur en écriture pour ajouter un prénom à la liste de prénoms de la personne.
- void addFirstnames (const **std : :vector**< **std : :string** > &newFirstnames)
  Accesseur en écriture pour ajouter une liste de prénoms à celle de la personne.
- void removeFirstname (const **std : :string** &oldFirstname)
  Accesseur en écriture pour supprimer un prénom de la liste des prénoms de la personne.
- void removeFirstnames (const **std : :vector**< **std : :string** > &oldFirstnames)
  Accesseur en écriture pour supprimer un ensemble de prénoms de la liste des prénoms de la personne.

Fonctions membres privées

- **std : :string** validateName (const **std : :string** &name) const
  Validation du nom.
- **std : :vector**< **std : :string** > validateFirstnames (const **std : :vector**< **std : :string** > &firstnames) const
  Validation de la liste des prénoms.

Attributs privés

- const Date birthdate_
  Date de naissance.
- Sex sex_
  Sexe.
- **std : :string** name_
  Nom de famille.
- **std : :vector**< **std : :string** > firstnames_
  Liste des prénoms.

6.2.1 Description détaillée

Classe représentant de manière extrêmement simplifiée une personne.

Les caractéristiques de la personne sont :
- sa date de naissance (*constante*) ;
- son sexe (*mutable*) ;
- son nom de famille (*mutable*) ;
- ses prénoms (*mutables*).

6.2.2 Documentation des constructeurs et destructeur

6.2.2.1 Person() [1/2]

```
nvs::Person::Person (
            const  std::vector<  std::string > & firstnames,
            const  std::string & name,
            Sex sex,
            const Date & birthdate )
```

Constructeur de Person.

Pour ce qui concerne les validations :
- la date de naissance est nécessairement valide ;
- le sexe n'est pas validé ;
- le nom de famille ne peut pas être vide ;
- seuls les prénoms non vide sont retenus, si un prénom apparaît plusieurs fois, seule sa première occurence est retenue.

Paramètres

firstnames	la liste ordonnée des prémons de la personne.
name	le nom de famille de la personne.
sex	le sexe de la personne.
birthdate	la date de naissance de la personne.

Exceptions

std : :invalid_argument	— name est vide ; — firstnames est vide ; — firstnames ne contient que des **std : :string** vides.

6.2.2.2 Person() [2/2]

```
nvs::Person::Person (
            const  std::vector<  std::string > & firstnames,
            const  std::string & name,
            Sex sex,
            int birthdateyear,
            unsigned birthdatemonth,
            unsigned birthdateday )
```

Constructeur de Person.

Pour ce qui concerne les validations :
- la date de naissance est validé à l'aide du constructeur de Date ;
- le sexe n'est pas validé ;
- le nom de famille ne peut pas être vide ;
- seuls les prénoms non vide sont retenus, si un prénom apparaît plusieurs fois, seule sa première occurence est retenue.

Paramètres

firstnames	la liste ordonnée des prémons de la personne.
name	le nom de famille de la personne.
sex	le sexe de la personne.
birthdateyear	l'année de naissance de la personne.
birthdatemonth	le mois de naissance de la personne.
birthdateday	le jour de naissance de la personne.

Exceptions

std : :invalid_argument	— date de naissance pas valide (voir le constructeur Date : :Date(int, unsigned, unsigned)) ; — name est vide ; — firstnames est vide ; — firstnames ne contient que des **std : :string** vides.

6.2.3 Documentation des fonctions membres

6.2.3.1 addFirstname()

```
void nvs::Person::addFirstname (
            const  std::string & newFirstname )
```

Accesseur en écriture pour ajouter un prénom à la liste de prénoms de la personne.

L'ajout se fait toujours en bout de la liste des prénoms.

Si le nouveau prénom est vide ou est un prénom déjà présent dans la liste, il n'est pas ajouté. Rien ne vient signaler cette absence d'ajout.

Paramètres

newFirstname	prénom à ajouter.

6.2.3.2 addFirstnames()

```
void nvs::Person::addFirstnames (
            const  std::vector< std::string > & newFirstnames )
```

Accesseur en écriture pour ajouter une liste de prénoms à celle de la personne.

L'ajout se fait toujours en bout de la liste des prénoms.

Si la liste de nouveaux prénoms est vide ou si chaque nouveau prénom est vide, aucun prénom n'est ajouté. Cela se passe silencieusement. Si un prénom de la liste des nouveaux prénoms est déjà présent dans la liste des prénoms de la personne, il n'est pas ajouté. Cela également se passe silencieusement. Si un prénom apparaît plus d'une fois dans la liste des nouveaux prénoms, il est ajouté, au plus, une seule fois.

Paramètres

newFirstnames	liste des prénoms à ajouter.

6.2.3.3 birthdate()

```
Date nvs::Person::birthdate ( ) const  [inline]
```

Accesseur en lecture de la date de naissance.

Remarque : la date de naissance est constante, il ne lui correspond donc pas d'accesseur en écriture.

Renvoie

la date de naissance de la personne.

6.2.3.4 firstname()

```
std::string nvs::Person::firstname (
            unsigned index = 0 ) const  [inline]
```

Accesseur en lecture d'un prénom en position d'index donné de la personne.

Paramètres

index	index du prénom désiré dans la liste de ceux-ci.

Renvoie

le prénom de la personne d'index fourni.

Exceptions

std : :out_of_range	index déborde du tableau de prénoms.

6.2.3.5 firstnames() [1/2]

```
std::vector< std::string > nvs::Person::firstnames ( ) const  [inline]
```

Accesseur en lecture des prénoms de la personne.

Renvoie

les prénoms de la personne.

6.2.3.6 firstnames() [2/2]

```
void nvs::Person::firstnames (
            const  std::vector<  std::string > & newFirstnames )
```

Accesseur en écriture pour modifier la liste des prénoms de la personne.

La liste des nouveaux prénoms doit respecter les mêmes contraintes que la liste des prénoms lors de la création d'une Person : elle ne peut être vide ni ne contenir que des prénoms vide. Si c'est le cas, la liste des prénoms n'est pas modifiée. Ceci se passe sans notification.

Si la liste des nouveaux prénoms contient plusieurs fois le même prénom, il n'est inclu qu'une seule fois à la liste des prénoms de la personne. Seuls les prénoms non vides sont pris en compte.

Paramètres

newFirstnames	nouvelle liste des prénoms de la personne.

6.2.3.7 firstnamesCount()

```
unsigned nvs::Person::firstnamesCount ( ) const  [inline]
```

Accesseur en lecture du nombre de prénoms de la personne.

Renvoie

le nombre de prénoms de la personne.

6.2.3.8 name() [1/2]

```
 std::string nvs::Person::name ( ) const  [inline]
```

Accesseur en lecture du nom de la personne.

Renvoie

le nom de famille de la personne.

6.2.3.9 name() [2/2]

```
void nvs::Person::name (
            const  std::string & newName )
```

Accesseur en écriture du nom de la personne.

Le nouveau nom est limité par la même contrainte que lors de la création d'une personne : il ne peut pas être vide. Par contre, aucune validation n'est réalisée quant à savoir si la nouvelle valeur du nom est réellement nouvelle.

Si le nouveau nom n'est pas valide, le nom de la personne reste inchangé. Rien ne vient signaler cette non-modification.

Paramètres

newName	nouveau nom de la personne.

6.2.3.10 removeFirstname()

```
void nvs::Person::removeFirstname (
            const  std::string & oldFirstname )
```

Accesseur en écriture pour supprimer un prénom de la liste des prénoms de la personne.

Si le prénom à supprimer n'est pas un prénom de la personne, rien ne se passe.

Paramètres

oldFirstname	prénom à supprimer de la liste des prénoms de la personne.

6.2.3.11 removeFirstnames()

```
void nvs::Person::removeFirstnames (
            const  std::vector< std::string > & oldFirstnames )
```

Accesseur en écriture pour supprimer un ensemble de prénoms de la liste des prénoms de la personne.

Si aucun des prénoms de la liste des prénoms à supprimer n'est un prénom de la personne, rien ne se passe.

Paramètres

oldFirstnames	liste des prénoms à supprimer de la liste des prénoms de la personne.

6.2.3.12 sex() [1/2]

```
Sex nvs::Person::sex ( ) const  [inline]
```

Accesseur en lecture du sexe de la personne.

Renvoie

la valeur du sexe de la personne.

6.2.3.13 sex() [2/2]

```
void nvs::Person::sex (
            Sex newSex )  [inline]
```

Accesseur en écriture du sexe de la personne.

Aucune validation n'est réalisée, y compris quant à savoir si la nouvelle valeur est réellement nouvelle.

Paramètres

newSex	nouvelle valeur du sexe de la personne.

6.2.3.14 validateFirstnames()

```
std::vector< std::string> nvs::Person::validateFirstnames (
            const  std::vector< std::string > & firstnames ) const  [private]
```

Validation de la liste des prénoms.

Une liste de prénoms valide contient au moins un prénom non vide. Si elle contient plusieurs prénoms, la multiplicité de chacun est de un.

Paramètres

firstnames	liste des prénoms à valider.

Renvoie

la liste des prénoms *validée* : chaque prénom y apparaît une seule fois, dans l'ordre de première occurence.

Exceptions

std : :invalid_argument	— firstnames est vide ; — firstnames ne contient que des **std : :string** vides.

6.2.3.15 validateName()

```
std::string nvs::Person::validateName (
            const  std::string & name ) const  [private]
```

Validation du nom.

Un nom valide est un nom non vide.

Paramètres

name	nom à valider.

Renvoie

le nom s'il est valide.

Exceptions

std : :invalid_argument	si name est vide.

La documentation de cette classe a été générée à partir du fichier suivant :
— person.h

7 Documentation des fichiers

7.1 Référence du fichier date.h

Définition de la classe nvs : :Date et de fonctions utilisant ce type.

```
#include <limits>
#include <array>
#include <string>
#include <ostream>
```

Classes

— class nvs : :Date
Classe représentant une date.

Espaces de nommage

— nvs
Espace de nom de Nicolas Vansteenkiste.

Fonctions

— **std : :ostream** & nvs : :operator<< (**std : :ostream** &out, const Date &in)
Opérateur d'injection d'une Date dans un flux en sortie.
— bool nvs : :operator< (const Date &lhs, const Date &rhs)
Opérateur de comparaison de deux dates.
— bool nvs : :operator== (const Date &lhs, const Date &rhs)
Opérateur de test d'égalité de deux dates.
— **std : :string** nvs : :to_string (const Date &in)
Fonction de conversion d'une nvs : :Date en ***std : :string****.*
— Date nvs : :today ()
Fonction retournant la date actuelle (au moment de l'exécution) sous la forme d'une Date.
— bool nvs : :leapYear (int year)
Fonction pour déterminer si une année est bissextile.

7.1.1 Description détaillée

Définition de la classe nvs : :Date et de fonctions utilisant ce type.

7.2 Référence du fichier formoption.h

Définition de l'énumération fortement typée nvs : :FormOption.

Espaces de nommage

— nvs
Espace de nom de Nicolas Vansteenkiste.

Énumérations

— enum nvs : :FormOption { nvs : :FormOption : :NONE = 0b0000, nvs : :FormOption : :SHORT = 0b0001, nvs : :FormOption : :LONG = 0b0010 }
Énumération pour spécifier les options de mise en forme d'impression ou de conversion binaire → chaîne de caractères.

7.2.1 Description détaillée

Définition de l'énumération fortement typée nvs : :FormOption.

7.3 Référence du fichier person.h

Définition de la classe nvs : :Person et de fonctions utilisant ce type.

```
#include "../date/date.h"
#include "../misc/formoption.h"
#include <string>
#include <vector>
#include <ostream>
```

Classes

— class nvs : :Person
Classe représentant de manière extrêmement simplifiée une personne.

Espaces de nommage

— nvs
Espace de nom de Nicolas Vansteenkiste.

Fonctions

— **std : :ostream** & nvs : :operator<< (**std : :ostream** &out, const Person &in)
Opérateur d'injection d'une Person dans un flux en sortie.
— **std : :string** nvs : :to_string (const Person &person, nvs : :FormOption fo=nvs : :FormOption : :SHORT)
*Fonction de conversion d'une Person en **std : :string**.*

7.3.1 Description détaillée

Définition de la classe nvs : :Person et de fonctions utilisant ce type.

7.4 Référence du fichier sex.h

Définition de l'énumération fortement typée nvs : :Sex et de diverses fonctions relatives.

```
#include <ostream>
#include <string>
#include "../misc/formoption.h"
```

Espaces de nommage

— nvs
Espace de nom de Nicolas Vansteenkiste.

Énumérations

— enum nvs : :Sex {
nvs : :Sex : :UNKNOWN, nvs : :Sex : :FEMALE, nvs : :Sex : :MALE, nvs : :Sex : :VOID,
nvs : :Sex : :OTHER }
Énumération fortement typée pour représenter le sexe d'une personne.

Fonctions

— **std : :ostream** & nvs : :operator<< (**std : :ostream** &out, Sex in)
Opérateur d'injection d'une variable ou valeur nvs : :Sex dans un flux en sortie.
— **std : :string** nvs : :to_string (Sex sex, FormOption fo=FormOption : :SHORT)
Fonction de conversion d'une variable / valeur nvs : :Sex en ***std : :string****.*

7.4.1 Description détaillée

Définition de l'énumération fortement typée nvs : :Sex et de diverses fonctions relatives.

7.5 Référence du fichier validation.hpp

Fonctions d'aide à la validation.

```
#include <functional>
#include <utility>
#include <string>
#include <stdexcept>
```

Espaces de nommage

— nvs
Espace de nom de Nicolas Vansteenkiste.

Fonctions

— template<typename T >
bool nvs : :between (const T &value, T min, T max)
Évaluation si une valeur est comprise entre deux autres.
— template<typename T >
std : :string nvs : :error_message (const T &value, T min, T max, const **std : :string** &msg="error : ")
Production d'un message d'erreur.
— template<typename T >
T nvs : :validate (const T &value, const T &min, const T &max, const **std : :string** &msg="error : ")
Validation d'une valeur entre deux bornes.

7.5.1 Description détaillée

Fonctions d'aide à la validation.

Index