# ConnectN

Generated by Doxygen 1.8.8

Thu Dec 11 2014 20:18:04

# Contents

# Chapter 1

# Deprecated List

**Member nvs::fromString (T &t, const std::string &s, bool iw=false)**

Depuis C++11, des fonctions d'extraction d'entiers, signés ou non, et de flottants depuis une chaîne sont disponibles. Il s'agit des fonctions `std::stoi`, `std::stol`, `std::stoll` et `std::stoul`, `std::stoull` et `std::stof`, `std::stod`, `std::stold`. Notez que les fonctions d'extraction d'entiers permettent de choisir la base, ce qui n'est pas de cas de nvs::fromString.

**Member nvs::fromString (const std::string &s, bool iw=false)**

Depuis C++11, des fonctions d'extraction d'entiers, signés ou non, et de flottants depuis une chaîne sont disponibles. Il s'agit des fonctions `std::stoi`, `std::stol`, `std::stoll` et `std::stoul`, `std::stoull` et `std::stof`, `std::stod`, `std::stold`. Notez que les fonctions d'extraction d'entiers permettent de choisir la base, ce qui n'est pas de cas de nvs::fromString.

**Member nvs::toString (const T &in)**

Depuis C++11, la fonction standard `to_string` offre la possibilité de convertir une valeur numérique en `std::string`.

# Chapter 2

# Namespace Index

## 2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

# Chapter 3

# Hierarchical Index

## 3.1 Class Hierarchy

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

# Chapter 4

# Class Index

## 4.1 Class List

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

# Chapter 5

# File Index

## 5.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 6

# Namespace Documentation

## 6.1 nvs Namespace Reference

Espace de nom de Nicolas Vansteenkiste.

### Classes

- class bad_string_convert

  *Classe d'exception utilisée lors des conversions depuis une `std::string`.*

### Functions

- template<typename T >
  T lineFromKbd (T &t, bool iw=false)

  *Lit toute une ligne au clavier et en extrait une seule donnée.*
- template<typename T >
  T lineFromKbd (bool iw=false)

  *Lit toute une ligne au clavier et en extrait une seule donnée.*
- template<typename T >
  T random_integer (T min=std::numeric_limits< T >::min(), T max=std::numeric_limits< T >::max())

  *Générateur d'entiers aléatoires.*
- template<typename T >
  std::string toString (const T &in)

  *Modèle de fonction de conversion d'un type quelconque vers une `string`.*
- template<typename T >
  T fromString (T &t, const std::string &s, bool iw=false)

  *Modèle de fonction de conversion d'une `string` vers un type quelconque sauf `char` et `char *`.*
- template<typename T >
  T fromString (const std::string &s, bool iw=false)

  *Modèle de fonction de conversion d'une `string` vers un type quelconque sauf `char` et `char *`.*
- char fromString (char &c, const std::string &s, bool iw=false)

  *Surcharge du modèle de fonction de conversion d'une `string` vers un type quelconque pour le type `char`.*
- char * fromString (char *, const std::string &, bool=false)=delete

  *Fonction mise en `delete` pour empêcher son existence.*
- template<>
  char fromString< char > (const std::string &s, bool iw)

  *Spécialisation du modèle de fonction de conversion d'une `string` vers un type quelconque pour le type `char`.*

- template<>
  char * fromString< char * > (const std::string &, bool)=delete

  *Spécialisation de modèle de fonction mise en* `delete` *pour empêcher son existence.*

### 6.1.1 Detailed Description

Espace de nom de Nicolas Vansteenkiste.

### 6.1.2 Function Documentation

#### 6.1.2.1 template<typename T > T nvs::fromString ( T & *t*, const std::string & *s*, bool *iw* = `false` )

Modèle de fonction de conversion d'une `string` vers un type quelconque *sauf* `char` et `char *`.

Le modèle de fonction utilise l'opérateur d'extraction d'un flux vers le type de retour. Celui-ci doit donc fournir un `operator>>` adéquat.

Lors de la conversion vers un booléen, seules les valeurs 0, pour `false`, et 1, pour `true`, sont acceptées.

**Deprecated** Depuis C++11, des fonctions d'extraction d'entiers, signés ou non, et de flottants depuis une chaîne sont disponibles. Il s'agit des fonctions `std::stoi`, `std::stol`, `std::stoll` et `std::stoul`, `std::stoull` et `std::stof`, `std::stod`, `std::stold`. Notez que les fonctions d'extraction d'entiers permettent de choisir la base, ce qui n'est pas de cas de nvs::fromString.

Pour une étude comparative des fonctions de conversion d'une `std::string` en `int`, allez voir `ici`.

**Parameters**

| | |
|---|---|
| *t* | Référence d'une variable qui accueille le résultat de la conversion de la string. |
| *s* | La `string` à convertir. |
| *iw* | Mis à `true`, les espaces blanches en début et en fin de la `string` sont ignorées ; par défaut ce paramètre est mis à `false` : les blancs ne sont pas ignorés. |

**Returns**

La représentation de la `string` dans le type demandé lors de l'appel.

**Exceptions**

| | |
|---|---|
| *nvs::bad_string_convert* | Outre les exceptions qui pourraient être lancées par l'opérateur d'extraction de flux, une nvs::bad_string_convert est levée si l'extraction du flux échoue ou si le flux n'est pas épuisé en fin d'extraction. |

#### 6.1.2.2 template<typename T > T nvs::fromString ( const std::string & *s*, bool *iw* = `false` )

Modèle de fonction de conversion d'une `string` vers un type quelconque *sauf* `char` et `char *`.

Le modèle de fonction utilise l'opérateur d'extraction d'un flux vers le type de retour. Celui-ci doit donc fournir un `operator>>` adéquat.

Lors de la conversion vers un booléen, seules les valeurs 0, pour `false`, et 1, pour `true`, sont acceptées.

**Deprecated** Depuis C++11, des fonctions d'extraction d'entiers, signés ou non, et de flottants depuis une chaîne sont disponibles. Il s'agit des fonctions `std::stoi`, `std::stol`, `std::stoll` et `std::stoul`, `std::stoull` et `std::stof`, `std::stod`, `std::stold`. Notez que les fonctions d'extraction d'entiers permettent de choisir la base, ce qui n'est pas de cas de nvs::fromString.

Pour une étude comparative des fonctions de conversion d'une `std::string` en `int`, allez voir `ici`.

Notez que ce modèle de fonction permet de construire des surcharges de fonctions ne différant entre elles que par leur type de retour !

**Parameters**

| | |
|---|---|
| *s* | La `string` à convertir. |
| *iw* | Mis à `true`, les espaces blanches en début et en fin de la `string` sont ignorées ; par défaut ce paramètre est mis à `false` : les blancs ne sont pas ignorés. |

**Returns**

La représentation de la `string` dans le type demandé lors de l'appel.

**Exceptions**

| | |
|---|---|
| *nvs::bad_string_convert* | Outre les exceptions qui pourraient être lancées par l'opérateur d'extraction de flux, une nvs::bad_string_convert est levée si l'extraction du flux échoue ou si le flux n'est pas épuisé en fin d'extraction. |

#### 6.1.2.3 char nvs::fromString ( char & *c*, const std::string & *s*, bool *iw* = `false` ) [inline]

Surcharge du modèle de fonction de conversion d'une `string` vers un type quelconque pour le type `char`.

On utilise une surcharge plutôt qu'une spécialisation de modèle. Les raisons en sont données `ici` et `ici`.

**Parameters**

| | |
|---|---|
| *c* | Référence de la variable qui accueille le résultat de la conversion de la string en caractère. |
| *s* | La `string` à convertir. |
| *iw* | Mis à `true`, les espaces blanches en début et en fin de la `string` sont ignorées ; par défaut ce paramètre est mis à `false` : les blancs ne sont pas ignorés. |

**Returns**

La représentation de la `string` sous la forme d'un `char`.

**Exceptions**

| | |
|---|---|
| *nvs::bad_string_convert* | Outre les exceptions qui pourraient être lancées par l'opérateur d'extraction de flux, une nvs::bad_string_convert est levée si l'extraction du flux échoue ou si le flux n'est pas épuisé en fin d'extraction. |

#### 6.1.2.4 char* nvs::fromString ( char *, const std::string & , bool = `false` ) [delete]

Fonction mise en `delete` pour empêcher son existence.

La conversion de `std::string` en `char *` avec nvs::fromString n'est pas sûre (cf. mémoire suffisamment allouée, pointeur ok) et / ou ne fonctionne pas. Comme je n'ai pas envie de développer la chose, je l'empêche avec `= delete`.

Pour passer d'une `std::string` à une chaîne *à la C* `char *`, il suffit d'utiliser les méthodes `c_str()` ou `data()` de `std::string`.

#### 6.1.2.5 template<> char* nvs::fromString< char * > ( const std::string & , bool ) [delete]

Spécialisation de modèle de fonction mise en `delete` pour empêcher son existence.

La conversion de `std::string` en `char *` avec nvs::fromString n'est pas sûre (cf. mémoire suffisamment allouée, pointeur ok) et / ou ne fonctionne pas. Comme je n'ai pas envie de développer la chose, je l'empêche avec `= delete`.

Pour passer d'une `std::string` à une chaîne *à la C* `char *`, il suffit d'utiliser les méthodes `c_str()` ou `data()` de `std::string`.

#### 6.1.2.6 template<> char nvs::fromString< char > ( const std::string & *s*, bool *iw* ) [inline]

Spécialisation du modèle de fonction de conversion d'une `string` vers un type quelconque pour le type `char`.

**Parameters**

| | |
|---|---|
| *s* | La `string` à convertir. |
| *iw* | Mis à `true`, les espaces blanches en début et en fin de la `string` sont ignorées ; par défaut ce paramètre est mis à `false` : les blancs ne sont pas ignorés. |

**Returns**

La représentation de la `string` sous la forme d'un `char`.

**Exceptions**

| | |
|---|---|
| *nvs::bad_string_convert* | Outre les exceptions qui pourraient être lancées par l'opérateur d'extraction de flux, une nvs::bad_string_convert est levée si l'extraction du flux échoue ou si le flux n'est pas épuisé en fin d'extraction. |

#### 6.1.2.7 template<typename T > T nvs::lineFromKbd ( T & *t*, bool *iw* = `false` )

Lit toute une ligne au clavier et en extrait une seule donnée.

La ligne lue est terminée par un `\n` qui est consommé mais pas pris en compte lors de la conversion vers la donnée binaire retournée.

Le modèle de fonction utilise nvs::fromString, donc l'opérateur d'extraction d'un flux vers le *template*. Celui-ci doit donc fournir un `operator>>` adéquat.

Lors de la lecture d'un booléen, seules les valeurs 0, pour `false`, et 1, pour `true`, sont acceptées.

**Parameters**

| | |
|---|---|
| *t* | Référence d'une variable qui accueille le résultat converti de la lecture. En cas de problème lors de la lecture le contenu de cette variable est indéterminé. |
| *iw* | Mis à `true`, les espaces blanches en début et en fin de la `string` sont ignorées ; par défaut ce paramètre est mis à `false` : les blancs ne sont pas ignorés. |

**Returns**

La donnée lue au clavier. En cas de problème lors de la lecture la valeur retournée est indéterminée.

**Exceptions**

| | |
|---|---|
| *nvs::bad_string_convert* | Outre les exceptions qui pourraient être lancées par l'opérateur d'extraction du flux, une nvs::bad_string_convert est levée si l'extraction du flux échoue ou si le flux n'est pas épuisé en fin d'extraction, c'est-à-dire si la donnée à extraire n'est pas seule sur la ligne lue, car nvs::fromString est utilisée. |

#### 6.1.2.8 template<typename T > T nvs::lineFromKbd ( bool *iw* = `false` )

Lit toute une ligne au clavier et en extrait une seule donnée.

La ligne lue est terminée par un `\n` qui est consommé mais pas pris en compte lors de la conversion vers la donnée binaire retournée.

Le modèle de fonction utilise nvs::fromString, donc l'opérateur d'extraction d'un flux vers le *template*. Celui-ci doit donc fournir un `operator>>` adéquat.

Lors de la lecture d'un booléen, seules les valeurs 0, pour `false`, et 1, pour `true`, sont acceptées.

Notez que ce modèle de fonction permet de construire des surcharges de fonctions ne différant entre elles que par leur type de retour !

**Parameters**

| | |
|---|---|
| *iw* | Mis à `true`, les espaces blanches en début et en fin de la `string` sont ignorées ; par défaut ce paramètre est mis à `false` : les blancs ne sont pas ignorés. |

**Returns**

La donnée lue au clavier. En cas de problème lors de la lecture la valeur retournée est indéterminée.

**Exceptions**

| | |
|---|---|
| *nvs::bad_string_convert* | Outre les exceptions qui pourraient être lancées par l'opérateur d'extraction du flux, une nvs::bad_string_convert est levée si l'extraction du flux échoue ou si le flux n'est pas épuisé en fin d'extraction, c'est-à-dire si la donnée à extraire n'est pas seule sur la ligne lue, car nvs::fromString est utilisée. |

#### 6.1.2.9 template<typename T > T nvs::random_integer ( T *min* = `std::numeric_limits<T>::min()`, T *max* = `std::numeric_limits<T>::max()` )

Générateur d'entiers aléatoires.

Il s'agit d'une distribution entière uniforme.

The effect is undefined if T is not one of : short, int, long, long long, unsigned short, unsigned int, unsigned long, or unsigned long long.

**Parameters**

| | |
|---|---|
| *min* | la valeur minimale pouvant être retournée. |
| *max* | la valeur maximale pouvant être retournée. |

**Returns**

un entier entre `min` et `max`.

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | si `min` > `max`. |

**Author**

nvs

**Version**

0.3

**Date**

2014

#### 6.1.2.10 template<typename T > std::string nvs::toString ( const T & *in* )

Modèle de fonction de conversion d'un type quelconque vers une `string`.

Pour que la fonction soit générée, le type de l'argument doit permettre son injection dans un flux en sortie à l'aide de l'opérateur <<.

**Deprecated** Depuis C++11, la fonction standard `to_string` offre la possibilité de convertir une valeur numérique en `std::string`.

**Parameters**

| | |
|---|---|
| *in* | La valeur à représenter sous la forme d'une string. |

**Returns**

La `string` représentant la valeur, sur base de l'`operator<<` de celle-ci.

# Chapter 7

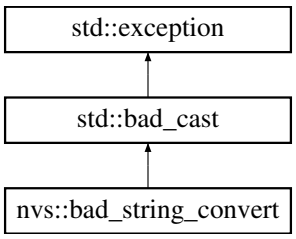
# Class Documentation

## 7.1 nvs::bad_string_convert Class Reference

Classe d'exception utilisée lors des conversions depuis une `std::string`.

`#include <stringConvert.hpp>`

Inheritance diagram for nvs::bad_string_convert:




### Public Member Functions

- virtual ~bad_string_convert ()=default

  *Destructeur par défaut.*
- virtual const char * what () const throw ()

  *Retourne une description de l'erreur rencontrée.*

### 7.1.1 Detailed Description

Classe d'exception utilisée lors des conversions depuis une `std::string`.

Levée par nvs::fromString.

### 7.1.2 Member Function Documentation

#### 7.1.2.1 virtual const char* nvs::bad_string_convert::what ( ) const throw ) [inline], [virtual]

Retourne une description de l'erreur rencontrée.

En fait, retourne toujours la chaîne de caractères : `nvs::bad_string_convert`. Pas très utile donc...

**Returns**

La chaîne de caractères : `nvs::bad_string_convert`.

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

- src/libs/stringConvert.hpp

## 7.2 ConnectN Class Reference

The ConnectN game.

`#include <ConnectN.h>`

### Public Types

- enum { DEFAULT_POWER = 4, DEFAULT_LINE = 6, DEFAULT_COLUMN = 7 }

### Public Member Functions

- ConnectN ()

  *Default ConnectN constructor.*
- ConnectN (unsigned power, unsigned line, unsigned column)

  *Custom ConnectN constructor.*
- void enroll (const Player *player)

  *Enroll a player.*
- void play (unsigned column)

  *Play at the given column.*
- unsigned power () const

  *Return the number of pieces to align.*
- unsigned line () const

  *Return the number of lines.*
- unsigned column () const

  *Return the number of columns.*
- bool started () const

  *Check if the game is started.*
- bool finished () const

  *Check if the game is finished.*
- const Player * winner () const

  *Return the winner.*
- const Player * activePlayer () const

  *Return the active player.*
- const std::array< std::pair < const Player *, Color >, 2 > players () const

  *Return an array of players, associated with their color.*
- Color color (const Player *) const

  *Return the color of the given player.*
- const std::vector< std::vector < Color > > & board () const

  *Return the game board.*

### Static Public Attributes

- static const unsigned MIN_POWER = 3

  *Minimum allowed power.*
- static const unsigned MAX_POWER = 10

  *Maximum allowed power.*
- static const unsigned DELTA_LINE = MAX_POWER + 10

  *Maximum line number depending on the maximum allowed power.*
- static const unsigned DELTA_COLUMN = MAX_POWER + 10

  *Maximum column number depending on the maximum allowed power.*

## 7.2.1 Detailed Description

The ConnectN game.

## 7.2.2 Member Enumeration Documentation

### 7.2.2.1 anonymous enum

**Enumerator**

***DEFAULT_POWER*** Default power.

***DEFAULT_LINE*** Default line.

***DEFAULT_COLUMN*** Default column.

## 7.2.3 Constructor & Destructor Documentation

### 7.2.3.1 ConnectN::ConnectN ( )

Default ConnectN constructor.

DEFAULT_POWER, DEFAULT_LINE and DEFAULT_COLUMN are used as default values.

### 7.2.3.2 ConnectN::ConnectN ( unsigned *power*, unsigned *line*, unsigned *column* )

Custom ConnectN constructor.

**Parameters**

| | |
|---|---|
| *power* | number of pieces to align |
| *line* | number of lines of the board |
| *column* | number of columns of the board |

## 7.2.4 Member Function Documentation

### 7.2.4.1 const Player* ConnectN::activePlayer ( ) const

Return the active player.

**Returns**

the active player if any, `nullptr` otherwise

#### 7.2.4.2 const std::vector<std::vector<Color> >& ConnectN::board ( ) const

Return the game board.

**Returns**

the game board

#### 7.2.4.3 Color ConnectN::color ( const Player * ) const

Return the color of the given player.

**Returns**

the color of the given player

#### 7.2.4.4 unsigned ConnectN::column ( ) const

Return the number of columns.

**Returns**

the number of columns

#### 7.2.4.5 void ConnectN::enroll ( const Player * *player* )

Enroll a player.

**Parameters**

| | |
|---|---|
| *player* | the player to enroll |

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | if the given player is already enrolled |
| *std::logic_error* | if two players are already enrolled |

#### 7.2.4.6 bool ConnectN::finished ( ) const

Check if the game is finished.

**Returns**

`true` if the game is finished, `false` otherwise

#### 7.2.4.7 unsigned ConnectN::line ( ) const

Return the number of lines.

**Returns**

the number of lines

#### 7.2.4.8 void ConnectN::play ( unsigned *column* )

Play at the given column.

This method tries to drop a piece in the column given as parameter. If it works, the board is checked to see if N pieces are aligned. If true, the winner is the current player; if not, the other player becomes the current player, and the game continues.

**Parameters**

| | |
|---|---|
| *column* | the column where to play |

**Exceptions**

| | |
|---|---|
| *std::logic_error* | if<br>• the game is not started<br>• the game is finished |
| *std::out_of_range* | if<br>• the column is full<br>• the column given as parameter is out of the board |

#### 7.2.4.9 const std::array<std::pair<const Player *, Color>, 2> ConnectN::players ( ) const

Return an array of players, associated with their color.

**Returns**

an array of players, associated with their color

#### 7.2.4.10 unsigned ConnectN::power ( ) const

Return the number of pieces to align.

**Returns**

the number of pieces to align

#### 7.2.4.11 bool ConnectN::started ( ) const

Check if the game is started.

**Returns**

`true` if the game is started, `false` otherwise

#### 7.2.4.12 const Player* ConnectN::winner ( ) const

Return the winner.

**Returns**

the winner if any, `nullptr` otherwise

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

- src/ConnectN.h
- src/ConnectN.cpp

## 7.3 Player Class Reference

A ConnectN player.

`#include <Player.h>`

### Public Member Functions

- Player (const std::string &name)
  *Player constructor.*
- const std::string & name () const
  *Return the name of the player.*

### 7.3.1 Detailed Description

A ConnectN player.

### 7.3.2 Constructor & Destructor Documentation

#### 7.3.2.1 Player::Player ( const std::string & *name* )

Player constructor.

**Parameters**

| | |
|---|---|
| *name* | the name of the player |

### 7.3.3 Member Function Documentation

#### 7.3.3.1 const std::string & Player::name ( ) const

Return the name of the player.

**Returns**

the name of the player

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

- src/Player.h
- src/Player.cpp

# Chapter 8

# File Documentation

## 8.1 src/Color.h File Reference

Color enum class definition.

```
#include <ostream>
#include <string>
```

### Enumerations

- enum Color { Color::NONE, Color::BLACK, Color::WHITE }

  *The Color enum class.*

### Functions

- std::string to_string (Color color)

  *to_string*
- std::ostream & operator<< (std::ostream &out, Color in)

  *Color output stream operator.*

### 8.1.1 Detailed Description

Color enum class definition.

### 8.1.2 Enumeration Type Documentation

#### 8.1.2.1 enum Color [strong]

The Color enum class.

**Enumerator**

***NONE*** an empty color

***BLACK*** a black piece

***WHITE*** a white piece

### 8.1.3 Function Documentation

#### 8.1.3.1 std::ostream& operator<< ( std::ostream & *out,* Color *in* )

Color output stream operator.

**Parameters**

| | |
|---|---|
| *out* | the output stream |
| *in* | the color |

**Returns**

the string representation of the color

#### 8.1.3.2 std::string to_string ( Color *color* )

to_string

**Parameters**

| | |
|---|---|
| *color* | the color |

**Returns**

the string representation of the color

## 8.2 src/ConnectN.h File Reference

ConnectN class definition.

```
#include <vector>
#include <array>
#include <string>
#include <sstream>
#include "Player.h"
#include "Color.h"
```

### Classes

- class ConnectN
  *The ConnectN game.*

### Functions

- std::string to_string (const ConnectN &in)
  *to_string*
- std::ostream & operator<< (std::ostream &out, ConnectN in)
  *ConnectN output stream operator.*

### 8.2.1 Detailed Description

ConnectN class definition.

### 8.2.2 Function Documentation

#### 8.2.2.1 std::ostream& operator<< ( std::ostream & *out*, ConnectN *in* )

ConnectN output stream operator.

**Parameters**

| | |
|---|---|
| *out* | the output stream |
| *in* | the ConnectN game |

**Returns**

the string representation of the ConnectN game

#### 8.2.2.2 std::string to_string ( const ConnectN & *in* )

to_string

**Parameters**

| | |
|---|---|
| *in* | |

**Returns**

## 8.3 src/libs/keyboard.hpp File Reference

Contient les modèles de fonctions pour lire des données au clavier.

```
#include "stringConvert.hpp"
#include <string>
#include <iostream>
```

### Namespaces

- nvs

  *Espace de nom de Nicolas Vansteenkiste.*

### Functions

- template<typename T >
  T nvs::lineFromKbd (T &t, bool iw=false)

  *Lit toute une ligne au clavier et en extrait une seule donnée.*
- template<typename T >
  T nvs::lineFromKbd (bool iw=false)

  *Lit toute une ligne au clavier et en extrait une seule donnée.*

### 8.3.1 Detailed Description

Contient les modèles de fonctions pour lire des données au clavier.

**Author**

nvs

## 8.4 src/libs/randomgenerator.hpp File Reference

Définitions de fonctions conviviales pour générer des séquences pseudo-aléatoires.

```
#include <random>
#include <limits>
#include <stdexcept>
#include <ctime>
```

### Namespaces

- nvs

  *Espace de nom de Nicolas Vansteenkiste.*

### Functions

- template<typename T >
  T nvs::random_integer (T min=std::numeric_limits< T >::min(), T max=std::numeric_limits< T >::max())

  *Générateur d'entiers aléatoires.*

### 8.4.1 Detailed Description

Définitions de fonctions conviviales pour générer des séquences pseudo-aléatoires.

## 8.5 src/libs/stringConvert.hpp File Reference

Contient les modèles de fonctions pour convertir depuis et vers une `string` standard.

```
#include <string>
#include <sstream>
#include <typeinfo>
```

### Classes

- class nvs::bad_string_convert

  *Classe d'exception utilisée lors des conversions depuis une `std::string`.*

### Namespaces

- nvs

  *Espace de nom de Nicolas Vansteenkiste.*

### Functions

- template<typename T >
  std::string nvs::toString (const T &in)

  *Modèle de fonction de conversion d'un type quelconque vers une `string`.*
- template<typename T >
  T nvs::fromString (T &t, const std::string &s, bool iw=false)

*Modèle de fonction de conversion d'une `string` vers un type quelconque sauf `char` et `char *`.*

- template<typename T >
  T nvs::fromString (const std::string &s, bool iw=false)

  *Modèle de fonction de conversion d'une `string` vers un type quelconque sauf `char` et `char *`.*
- char nvs::fromString (char &c, const std::string &s, bool iw=false)

  *Surcharge du modèle de fonction de conversion d'une `string` vers un type quelconque pour le type `char`.*
- char * nvs::fromString (char *, const std::string &, bool=false)=delete

  *Fonction mise en `delete` pour empêcher son existence.*
- template<>
  char nvs::fromString< char > (const std::string &s, bool iw)

  *Spécialisation du modèle de fonction de conversion d'une `string` vers un type quelconque pour le type `char`.*
- template<>
  char * nvs::fromString< char * > (const std::string &, bool)=delete

  *Spécialisation de modèle de fonction mise en `delete` pour empêcher son existence.*

### 8.5.1 Detailed Description

Contient les modèles de fonctions pour convertir depuis et vers une `string` standard.

**Author**

nvs

## 8.6 src/Player.h File Reference

Player class definition.

```
#include <string>
#include "Color.h"
```

### Classes

- class Player

  *A ConnectN player.*

### 8.6.1 Detailed Description

Player class definition.

# Index