

Initiation à Doxygen pour C et C++



Doxygen est un outil qui peut s'avérer très pratique, notamment dans des développements de gros projets et autres comme des bibliothèques de fonctions, pour en créer des documentations techniques pour d'autres développeurs !

Avec un peu d'effort, de tests et quelques petites touches personnelles, on peut arriver à des résultats dignes des documentations faites par des professionnelles dans l'industrie des logiciels.


- I. [Introduction](#)
- II. [Les blocs de documentation et les balises standards](#)
- III. [Mise en place de la documentation](#)
- IV. [Avant d'aller plus loin...](#)
- V. [Configuration de Doxygen avec Doxywizard](#)
- VI. [Configuration avancée](#)
- V. [Bibliographie](#)

I. Introduction

À l'image de Javadoc, l'outil d'auto-documentation pour Java, Doxygen permet de créer des documentations techniques pour notamment le C et le C++ mais couvre également d'autres langages, y compris Java !

Ce qui sera étudié dans ce tutoriel, c'est l'utilisation basique de Doxygen avec un petit détournement vers des fonctions avancées pour générer des documentations de références comme par exemple celles de GTK+.

II. Les blocs de documentation et les balises standards

Ce chapitre vous présente succinctement les balises les plus utilisées avec Doxygen. Pour la liste complète, rendez-vous sur le site officiel:  [Special Commands](#). De plus, la portée d'une balise dépend essentiellement de deux choses:

- Une nouvelle balise est rencontrée et est maintenant considérée.
- Une ligne vide est laissée dans le bloc des commentaires.

II-A. Les blocs de documentation

Diverses combinaisons sont possibles pour créer des blocs de documentation, dans le style C ou C++ pour notre cas !

Style C avec deux *

```
/**  
 * ... Documentation ...  
 */
```

Style C avec un !

```
/*!  
 * ... Documentation ...  
 */
```

Il est important de noter qu'il n'est pas nécessaire de mettre des astérisques au début de chaque nouvelle ligne et ceux qui utilisent Eclipse avec le CDT, celles-ci sont ajoutées automatiquement et permettent, croyons-nous,

Style C++ avec trois /

```

///
/// ... Documentation ...
///

Style C++ avec un !

//!
//! ... Documentation ...
//!

```

II-B. \file

Permet de créer un bloc de documentation pour un fichier source ou d'en-tête.

Déclaration

```
\file [<name>]
```

Exemple

```
\file main.c
```

II-C. \brief

Permet de créer une courte description dans un bloc de documentation. La description peut se faire sur une seule ligne ou plusieurs.

Déclaration

```
\brief {brief description}
```

Exemple : description sur une seule ligne

```
\brief Courte description.
```

Exemple : description sur plusieurs lignes

```
\brief Courte description.
    Suite de la courte description.
```

Une description brève peut être suivie par une description détaillée, celle-ci peut être écrite sans utiliser de balise. Dans ce cas il faut alors être prudent pour éviter les conflits entre les deux. Dans le cas qui s'applique à nous, on peut utiliser un saut de ligne pour distinguer les deux :

```

/** \file Fonctions.h
 * \brief Description brève
 *
 * Description détaillée
 */

```

Si nous voulons uniquement une détaillée il suffit de ne pas mettre la balise \brief.

II-D. \author

Permet d'ajouter un nom d'auteur (ex: l'auteur d'un fichier ou d'une fonction). Plusieurs balises peuvent être présentes, lors de la génération un seul paragraphe **Auteur** sera créé mais les listes d'auteurs seront séparées une ligne vide. Une seule balise peut accueillir plusieurs auteurs.

Déclaration

```
\author { list of authors }
```

Exemple

```
\author Franck.H
```

II-E. \version

Permet l'ajout du numéro de version (ex: dans le bloc de documentation d'un fichier).

Déclaration

```
\version { version number }
```

Exemple

```
\version 0.1
```

II-F. \date

Permet l'ajout d'une date. Plusieurs balises peuvent être présentes, lors de la génération un seul paragraphe **Date** sera créé mais chaque date sera séparée par une ligne vide. Une seule balise peut accueillir plusieurs dates.

Déclaration

```
\date { date description }
```

Exemple

```
\date 10 septembre 2007
```

II-G. \struct

Permet la création d'un bloc de documentation pour une structure. Cette balise peut prendre jusqu'à trois arguments qui sont dans l'ordre :

1. Nom de la structure
2. Nom du fichier d'en-tête (ex: le fichier où elle est définie)
3. Nom optionnel pour masquer le nom affiché par le second argument

Si le second argument est fourni, un lien HTML vers le code source du fichier d'en-tête spécifié sera créé. Le dernier argument permet quant à lui de simplement changer éventuellement le nom de ce lien qui par défaut est le nom du fichier fournit en second argument.

Déclaration

```
\struct <name> [<header-file>] [<header-name>]
```

Exemple

```
\struct Str_t
```

Exemple

```
\struct Str_t str.h "Définition"
```

Chacune des variables à l'intérieur de la struct devra être expliquée. Vous pouvez noter par exemple que ce ne sont pas toutes les variables qui ont une description brève et une description détaillée, il faut aller suivant le cas et la nature de la variable qui est à commenter :

```
/** \struct monStruct
 * \brief Description brève
 *
 * Description détaillée
 */

typedef struct
{
    int param1;      /** < \brief Description brève */
    float param2;    /** < \brief Description brève
                     *
                     * Description détaillée
                     */
    char* param3;    /** < Description détaillée */
} monStruct
```

II-H. \enum

Permet la création d'un bloc de documentation pour une énumération de constantes.

Déclaration

```
\enum <name>
```

Exemple

```
\enum Str_err_e
```

La façon de commenter un enum est donc similaire à celle utilisé pour commenter un struct à la différence que les éléments ne peuvent avoir une description brève et une détaillée. Voici donc un exemple de commentaire d'une énumération:

```
/** \enum monEnumeration
 * \brief Description brève
 *
 * Description détaillée
 */

typedef enum {
    elem1, /**< Description de elem1*/
    elem2, /**< Description de elem2*/
    elem3 /**< Description de elem3*/
} monEnumeration;
```

II-I. \union

Permet la création d'un bloc de documentation pour une union. L'utilisation est la même que pour une structure (*voir le chapitre II-G*).

Déclaration

```
\union <name> [<header-file>] [<header-name>]
```

II-J. \fn

Permet la création d'un bloc de documentation pour une fonction ou méthode.

Déclaration

```
\fn (function declaration)
```

Exemple

```
\fn int main (void)
```



Il est possible d'omettre cette balise lorsque le bloc de documentation commence juste au-dessus de la déclaration/définition de la fonction/méthode.

Doxygen ajoutera de lui-même la signature de la fonction ou de la méthode. On peut donc en conclure que nous pouvons documenter une fonction

à peu près n'importe où dans le code mais dans ce cas, il faut ajouter la balise !

II-K. \def

Permet la création d'un bloc de documentation pour une macro ou constante symbolique.

Déclaration

```
\def <name>
```

Exemple

```
\def MAX(x,y)
```

Exemple

```
/** \def MON_NOM_DE_CONSTANTE
 * \brief Description brève
 *
 * Description détaillée.
```

```
* /
```

```
#define MON_NOM_DE_CONSTANTE 0
```

II-L. \param

Permet d'ajouter un paramètre dans un bloc de documentation d'une fonction ou d'une macro. Le premier paramètre est le nom de l'argument à documenter et le second le bloc de description le concernant.

Déclaration

```
\param <parameter-name> { parameter description }
```

Exemple

```
\param self Pointeur sur un objet de type Str_t.
```

Cette balise permet aussi d'ajouter en option un tag pour montrer qu'il s'agit d'un argument entrant, sortant ou les deux en précisant au choix: **[in]**, **[out]** ou **[in, out]** de cette manière:

Exemple avec un tag [in]

```
\param[in] self Pointeur sur un objet de type Str_t.
```

La sortie serait alors:

Sortie

Paramètres:

```
[in] self Pointeur sur un objet de type Str_t.
```

II-M. \return

Permet de décrire le retour d'une fonction ou d'une méthode.

Déclaration

```
\return { description of the return value }
```

Exemple

```
\return Instance de l'objet, NULL en cas d'erreur.
```

II-N. \bug

Permet de commencer un paragraphe décrivant un bug (ou plusieurs). L'utilisation est identique à la balise **\author** (voir la section [II-D](#)).

Déclaration

```
\bug { bug description }
```

Exemple

```
\bug Problème d'affichage du texte en sortie
```

II-O. \deprecated

Permet d'ajouter un paragraphe précisant que la fonction, marco, etc... est dépréciée, qu'il ne faut donc plus l'utiliser.

Déclaration

```
\deprecated { description }
```

Exemple

`\deprecated` Fonction dépréciée, ne plus utiliser !

II-P. `\class`

Permet de créer un bloc de documentation d'une classe (C++). L'utilisation est identique à la balise `\struct` (voir la section [II-G](#)).

Déclaration

```
\class <name> [<header-file>] [<header-name>]
```

Exemple

```
\class Str
```

II-Q. `\namespace`

Permet de créer un bloc de documentation pour un espace de nom (C++).

Déclaration

```
\namespace <name>
```

Exemple

```
\namespace std
```

II-R. Listes et groupes

Il peut être intéressant dans certaines occasions de pouvoir faire une liste d'éléments pour permettre une meilleure lisibilité du texte ou tout simplement car la situation l'exige. Pour ce faire, nous avons à notre disposition deux « tag », - et -#. Le premier génère une liste ayant des points précédents les éléments, le second inscrivant des chiffres devant. Voici donc un exemple d'une liste que l'on pourrait retrouver dans la description détaillée d'une fonction, d'un struct et etc.

```
* .... La partie avant la liste ....
*
* Voici une liste
* - Élément 1
*   - Sous élément 1
*     - sous sous élément 1
*       - sous sous élément 2\n
*         Texte du sous sous élément 2
*   .
*   Texte du sous élément 1
* - Sous élément 2
*   -# Sous sous élément 1\n
*     Texte du sous sous élément 1
*   -# Sous sous élément 2
* - Élément 2\n
*   Texte de l'élément deux
*
* .... La partie après la liste ....
```

Faire une liste

On peut voir qu'il suffit de décaler de un caractère pour pouvoir créer une sous liste en inscrivant un des deux « tags » au début de la ligne. De plus, on utilise ici le point sur une ligne vide pour indiquer la fin d'une liste. S'il ne se trouvait pas là, le texte « Texte du sous élément1 » serait à la suite de « Texte du sous sous élément 2 ». Dans l'exemple nous employons aussi l'instruction *n*. L'effet de celle-ci sera expliqué plus en détail ultérieurement. Voici maintenant ce que l'exemple donnerait:

Voici une liste

- Élément 1
 - 1. Sous élément 1
 - sous sous élément 1

- sous sous élément 2
 - Texte du sous sous élément 2
- Texte du sous élément 1

2. Sous élément 2

1. Sous sous élément 1
 - Texte du sous sous élément 1
2. Sous sous élément 2

- Élément 2
 - Texte de l'élément deux

Comment faire un groupe et sous-groupe

Doxygen nous permet aussi de regrouper certains éléments de notre programme entre eux comme on le désire. Par exemple, on pourrait vouloir regrouper ensemble toutes les fonctions qui interagissent avec un struct donné, ou on pourrait regrouper tous les struct de notre programme sous un groupe modèle d'implantation. Ceci facilite alors la navigation dans la documentation tout en donnant une meilleure idée de ce que font les éléments. Pour gérer les groupes nous nous servons essentiellement de deux instructions, soit *defgroup* et *addtogroup*. La première permet de créer un nouveau groupe et la seconde d'ajouter des éléments à celui-ci. Cependant, il n'est pas nécessaire de toujours utiliser *addtogroup* pour ajouter chacun des éléments au groupe. Par exemple, si nous créons un groupe, fonction1, fonction2 et fonction3 appartiennent au groupe ID_DU_GROUPE car elles sont situées entre les deux tags.

```
/** \defgroup ID_DU_GROUPE Description du groupe*/
*{
/* Commentaires de la fonctions*/
void fonction1(int param1,int param2,int* err);

/* Commentaires de la fonctions*/
void fonction2(int param1,int param2,int* err);

/* Commentaires de la fonctions*/
void fonction3(int param1,int param2,int* err);
*}

```

Exemple de base d'un groupe

De la même façon si l'on ajoute par la suite des éléments au groupe nous aurons monStruct et monEnum qui appartiennent aussi au groupe ID_DU_GROUPE.

```
/** \addtogroup ID_DU_GROUPE */
*{
/* Commentaires du struct*/
typedef struct
{
    /* Paramètres su struct*/
} monStruct;

/* Commentaires de l'énumération*/
typedef enum {
    /* Les constantes de l'énumération*/
} monEnum;

*}

```

Exemple d'ajout à un groupe

De plus, nous pouvons créer des sous-groupes. Pour se faire il suffit de se créer un nouveau groupe à l'intérieur d'un groupe déjà existant. Voici un exemple où nous avons décalé les groupes pour que vous puissiez mieux voir ce qu'il faut faire.

```
/** \defgroup groupe1 Mon groupe principal */
*{

    /** \defgroup groupe2 Sous-groupe du groupe principal*/
    *{
    *}

*}

```

Exemple d'un sous-groupe

Finalement, une fois le « compiler » vous allez trouver tous les groupes sous la section modules dans le document html.

 Pour plus d'informations pour créer des listes !

II-S. Autres balises utiles

`\n`

Dans dOxygen même si vous changez de ligne le texte va être inscrit à la suite du texte de la ligne précédente. Ou si vous laissez une ligne vide cela va être pris comme le début d'un nouveau paragraphe. Dans ce cas, il faut utiliser l'instruction `\n` qui permettra de passer à la ligne suivante.

`\sa`

Il peut être intéressant de faire référence à certaines autres parties de la documentation. En utilisant cette instruction on créera un paragraphe intitulé

Voir également: comprenant les références que l'on désire y mettre. Supposons que nous ayons un struct nommée `monStruct`. Si on écrit « `\sa monStruct` », dOxygen créera automatiquement un lien vers les informations concernant `monStruct`. Plus globalement, un lien sera automatiquement créé à chaque fois que dOxygen reconnaîtra le nom d'un élément commenté.

`\note`

Si l'on veut ajouter une note à la documentation nous pouvons utiliser cette instruction. **Note:** en gras sera inscrit au début de la phrase puis suivra ce que vous avez inscrit après.

`\warning`

Ici on a un paragraphe intitulé **Avertissement:** sera généré et vous pourrez inscrire l'énoncé de l'avertissement que vous désirez.

`\mainpage`

Quand vous ouvrez la page `index.html` vous avez devant vous une page pour ainsi dire vide. L'instruction `\mainpage` vous permet d'inscrire quelque chose à la page principale. Par exemple une description de votre programme ainsi que la liste des personnes ayant participé à sa réalisation. Pour ce faire vous devez créer un bloc de commentaire débutant par la dite instruction dans n'importe quel fichier de votre programme et inscrire par la suite ce que vous voulez qu'il apparaisse à la première page.

`\todo`

Si on le désire, on peut utiliser dOxygen pour générer la liste des choses à faire. Il suffit d'ajouter l'instruction `\todo` dans un bloc de commentaire de d'inscrire par la suite ce qu'il reste à faire. Une nouvelle page sera alors créée sous la rubrique pages associées. Elle contiendra la description de la tâche que vous venez d'écrire ainsi que son emplacement.

III. Mise en place de la documentation

III-A. Informations d'en-tête

Nous allons voir ici une manière de mettre un bloc d'informations d'en-tête d'un fichier avec les numéros de versions, auteurs, nom de fichiers, etc...

```
/**
 * \file main.c
 * \brief Programme de tests.
 * \author Franck.H
 * \version 0.1
 * \date 11 septembre 2007
 *
 * Programme de test pour l'objet de gestion des chaînes de caractères Str_t.
 *
 */
```


L'ordre des balises n'a que très peu d'importance mais cela a un impact sur l'ordre de génération du paragraphe et donc de l'affichage. Ce qu'on peut remarquer de plus et qui n'a pas été abordé jusque-là, c'est qu'on peut ajouter des commentaires en-dehors de la balise `\brief`. Ceci sera en effet considéré par Doxygen comme une description détaillée et se trouvera alors dans un paragraphe intitulé **Description détaillée**.

On peut également voir que dans le champ **date**, la date peut prendre la forme que l'on souhaite. La balise sert surtout pour créer le paragraphe avec le bon titre.

III-B. Documentation d'une fonction/méthode

Que ce soit pour une fonction (C) ou une méthode de classes (C++), ce bloc est identique. Il faut noter qu'il doit y avoir une balise `\param` par argument de la fonction.

Tous les paramètres seront alors dans un même paragraphe **Paramètres** !

```
/**
 * \fn static Str_t * str_new (const char * sz)
 * \brief Fonction de création d'une nouvelle instance d'un objet Str_t.
 *
 * \param l'entrée sz est la chaîne à stocker dans l'objet Str_t, ne peut être NULL.
 * \return Instance nouvellement allouée d'un objet de type Str_t ou NULL.
 */
```



Remarquez que pour la balise `\fn`, il faut fournir le prototype complet de la fonction ou de la méthode. De plus, dans les commentaires qui suivent la balise `\param` il faut préciser les entrées à la fonction et les paramètres qui servent aux effets de bord toujours de la fonction (`\param` regroupe donc les commentaires entrées et résultats suivant nos commentaires spécialisés vus dans le cours.



Il ne faut surtout pas oublier de bien fournir la balise `\param` car une vérification syntaxique de celle-ci sera faite par Doxygen et au moindre problème, il ne générera pas votre documentation !

III-C. Documentation d'une structure/union

La documentation d'une structure et de d'une union se fait de la même manière, nous allons donc voir ici que le cas d'une structure, ce qui sera retranscrit dans les exemples complets plus bas.

```
/**
 * \struct Str_t
 * \brief Objet chaîne de caractères.
 *
 * Str_t est un petit objet de gestion de chaînes de caractères.
 * La chaîne se termine obligatoirement par un zéro de fin et l'objet
 * connaît la taille de chaîne contient !
 */
```

Jusque-là rien de bien difficile, on peut simplement remarquer que nous n'avons ici, pas renseigné les champs optionnels de la balise `\struct`. Une particularité qui n'a pas été abordée, est que nous pouvons commenter les différents champs d'une structure, d'une union et même des variables membres d'une classe.

Voici comment procéder:

```
typedef enum
{
    STR_NO_ERR,      /*!< Pas d'erreur. */
    STR_EMPTY_ERR,  /*!< Erreur: Objet vide ou non initialisé. */

    NB_STR_ERR      /*!< Nombre total de constantes d'erreur. */
}
Str_err_e;
```

L'opérateur qu'il faut essentiellement retenir est "<" qui permet alors de documenter un membre, ici d'une structure, ce qui produit une génération de ce genre:

Exemple de sortie

III-D. Exemple

Voici un programme écrit dans un seul fichier et commenté avec les balises de Doxygen :

```
/**
 * \file main.c
 * \brief Programme de tests.
 * \author Franck.H
 * \version 0.1
 * \date 6 septembre 2007
 *
 * Programme de test pour l'objet de gestion des chaînes de caractères Str_t.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/**
 * \struct Str_t
 * \brief Objet chaîne de caractères.
 *
 * Str_t est un petit objet de gestion de chaînes de caractères.
 * La chaîne se termine obligatoirement par un zéro de fin et l'objet
 * connait la taille de chaîne contient !
 */
typedef struct
{
    char * sz; /*!< Chaîne avec caractère null de fin de chaîne. */
    size_t len; /*!< Taille de la chaîne sz sans compter le zéro de fin. */
}
Str_t;

/**
 * \enum Str_err_e
 * \brief Constantes d'erreurs.
 *
 * Str_err_e est une série de constantes prédéfinie pour diverses futures
 * fonctions de l'objet Str_t.
 */
typedef enum
{
    STR_NO_ERR, /*!< Pas d'erreur. */
    STR_EMPTY_ERR, /*!< Erreur: Objet vide ou non initialisé. */

    NB_STR_ERR /*!< Nombre total de constantes d'erreur. */
}
Str_err_e;

/**
 * \fn static Str_err_e str_destroy (Str_t ** self)
 * \brief Fonction de destruction de l'objet Str_t.
 *
 * \param l'entrée self qui ne peut être NULL, double indirection de ll'objet Str_t à détruire. *self, contiendr:
 * \return STR_NO_ERR si aucune erreur, STR_EMPTY_ERR sinon.
 */
static Str_err_e str_destroy (Str_t ** self)
{
    Str_err_e err = STR_EMPTY_ERR;

    if (self != NULL && *self != NULL)
    {
        free (* self);
        *self = NULL;

        err = STR_NO_ERR;
    }

    return err;
}

/**
 * \fn static Str_t * str_new (const char * sz)
 * \brief Fonction de création d'une nouvelle instance d'un objet Str_t.
```

```

*
* \param l'entrée sz est la chaîne à stocker dans l'objet Str_t, ne peut être NULL.
* \return Instance nouvelle allouée d'un objet de type Str_t ou NULL.
*/
static Str_t * str_new (const char * sz)
{
    Str_t * self = NULL;

    if (sz != NULL && strlen (sz) > 0)
    {
        self = malloc (sizeof (* self));

        if (self != NULL)
        {
            self->len = strlen (sz);
            self->sz = malloc (self->len + 1);

            if (self->sz != NULL)
            {
                strcpy (self->sz, sz);
            }
            else
            {
                str_destroy (& self);
            }
        }
    }

    return self;
}

/**
* \fn int main (void)
* \brief Entrée du programme.
*
* \return EXIT_SUCCESS - Arrêt normal du programme.
*/
int main (void)
{
    Str_err_e err;
    Str_t * my_str = str_new ("Ma chaîne de caracteres !");

    if (my_str != NULL)
    {
        printf ("%s\n", my_str->sz);
        printf ("Taille de la chaîne : %d\n", my_str->len);

        err = str_destroy (& my_str);

        if (! err)
        {
            printf ("L'objet a été libéré correctement !\n");
        }
    }

    return EXIT_SUCCESS;
}


```

IV. Avant d'aller plus loin...

Si vous n'avez pas encore Doxygen d'installé, il va falloir franchir le pas maintenant. Vous pouvez trouver différentes archives et installateurs pour divers systèmes sur cette page : <http://www.stack.nl/~dimitri/doxygen/download.html#latestsrc>

Si vous avez par exemple une distribution comme Debian, vous pouvez passer par les dépôts mais il vous faut (*par rapport à l'installateur pour la version Windows qui inclut le tout*) alors installer plusieurs programmes :

```
sudo aptitude install graphviz doxygen doxygen-gui
```

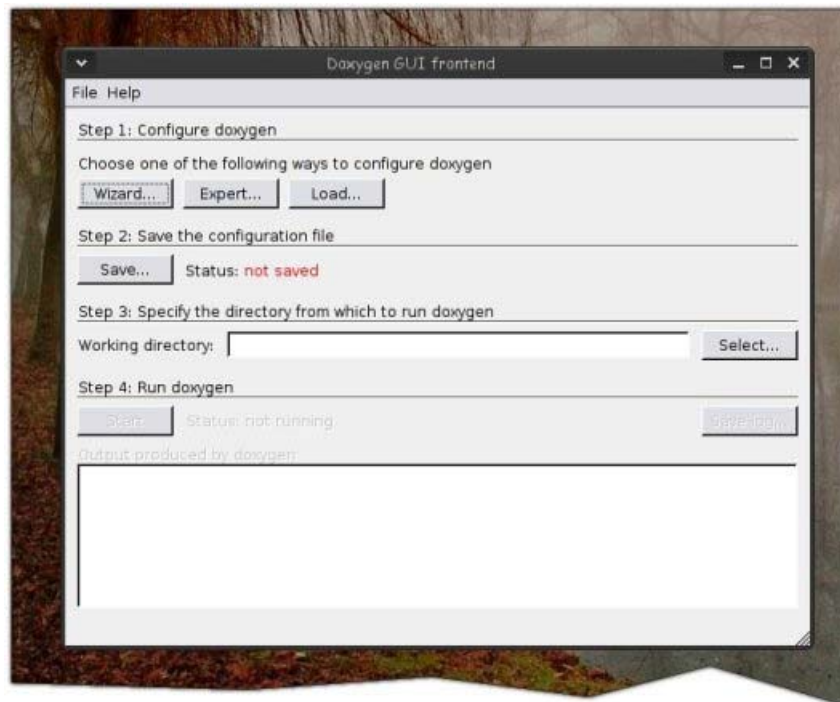
Dont  **graphviz** permet la génération de graphiques (*entre autres, des graphiques de dépendance*) et **doxygen-gui** est l'interface graphique (*doxywizard*) !

V. Configuration de Doxygen avec Doxywizard

Allons-y... lancez Doxygen ! Sous Linux par la console vous devez appeler le programme nommé **doxywizard** :

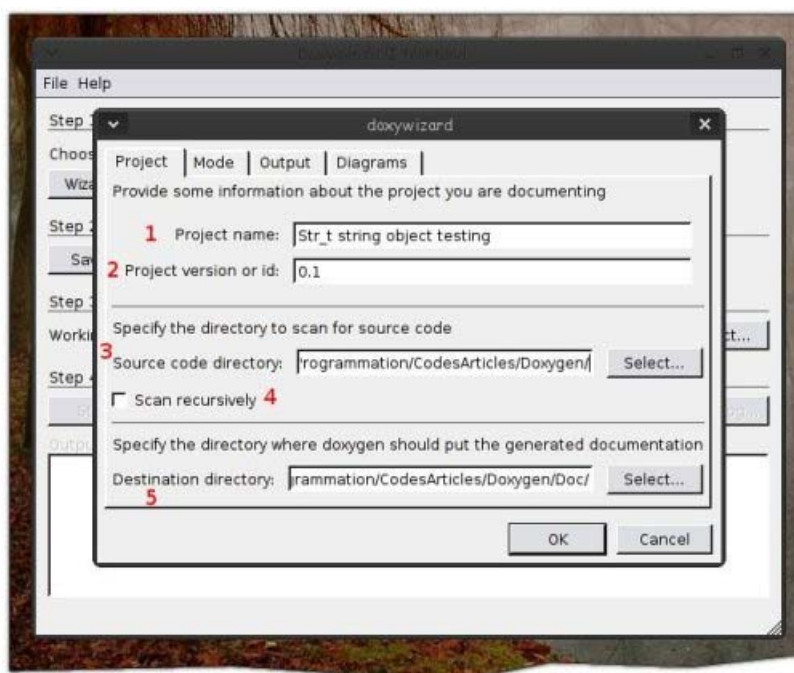
```
franhec@lucie:~$ doxywizard
```

Vous devriez vous retrouver devant une interface semblable à celle-ci :



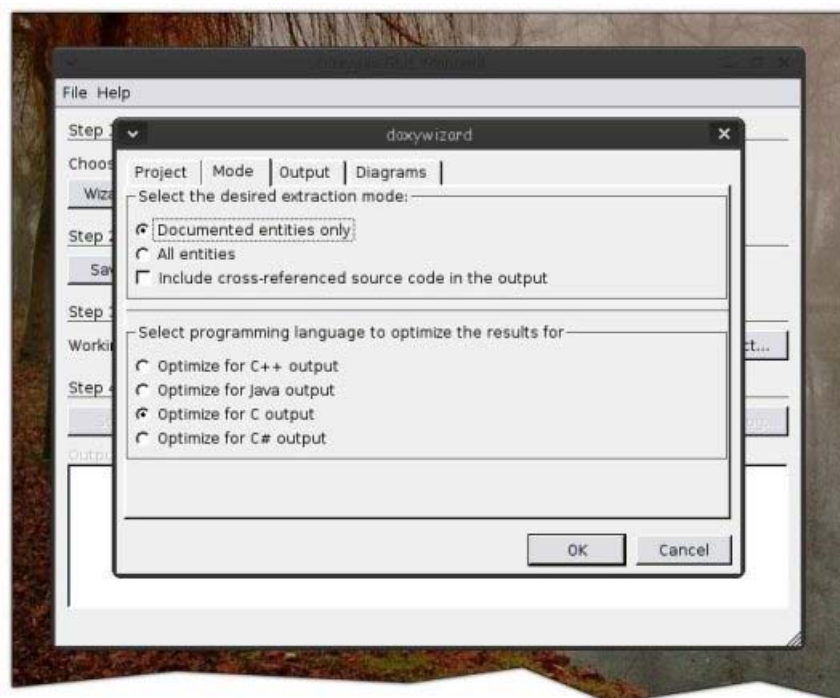
Nous allons maintenant voir comment configurer un projet. Pour commencer, nous passons par l'assistant donc le bouton **Wizard...**

V-A. Onglet "Project"



1. Le nom de votre projet.
2. La version du projet.
3. Le répertoire du code source dont il faut générer la documentation.
4. Si cette case est cochée, Doxygen ira fouiller également dans les sous-répertoires à la recherche de codes sources.
5. Emplacement où il faut générer la documentation. Un répertoire **html** est créé par défaut par Doxygen à l'emplacement spécifié où la documentation y sera placée.

V-B. Onglet "Mode"



Dans ce second onglet d'options, dans la première série nous pouvons déterminer le détail du contenu (*plus ou moins exactement*). Par défaut l'option est sur

Documented entities only, ceci permettra simplement la génération de la documentation des parties documentées, les autres ne seront pas référencées.

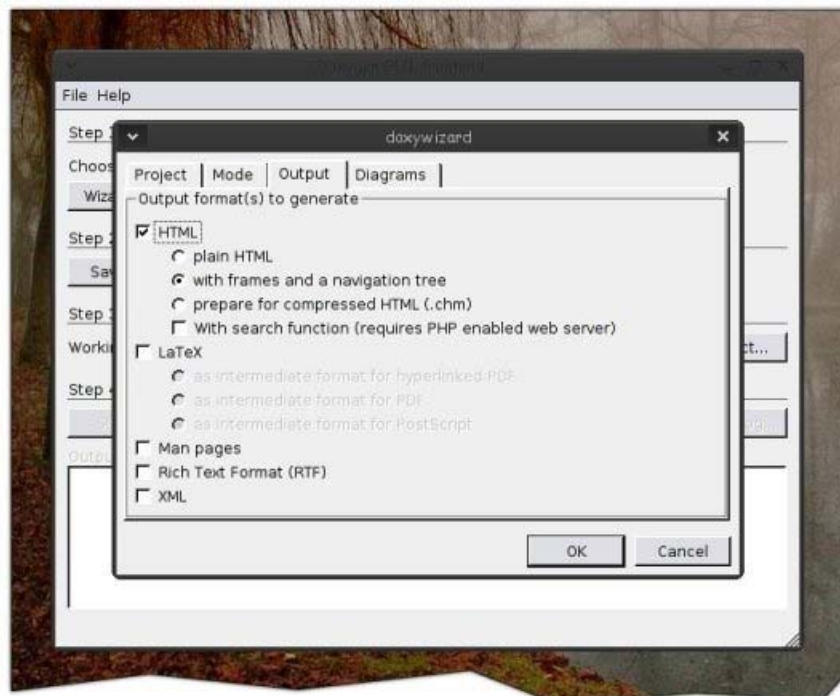
La seconde option, permet justement de faire référencer la totalité du code, donc les parties **private**, **static**, etc...

En effet, même si vous créez de la documentation pour une fonction privée par exemple, celle-ci ne sera pas ajoutée dans la génération si vous utilisez la première option mais, la seconde fait tout générer, ce que vous ne voulez peut-être pas, nous verrons la façon de régler cette partie avec précision dans le prochain chapitre !

La case à cocher "*Include cross-referenced source code in the output*" permet l'inclusion du code source complet dans la documentation tout en ajoutant des liens hypertext pour chaque partie documentée entre la documentation et le code !

La seconde série d'options permet de définir la façon dont le code sera colorisé et présenté, donc par rapport à votre langage !

V-C. Onglet "Output"

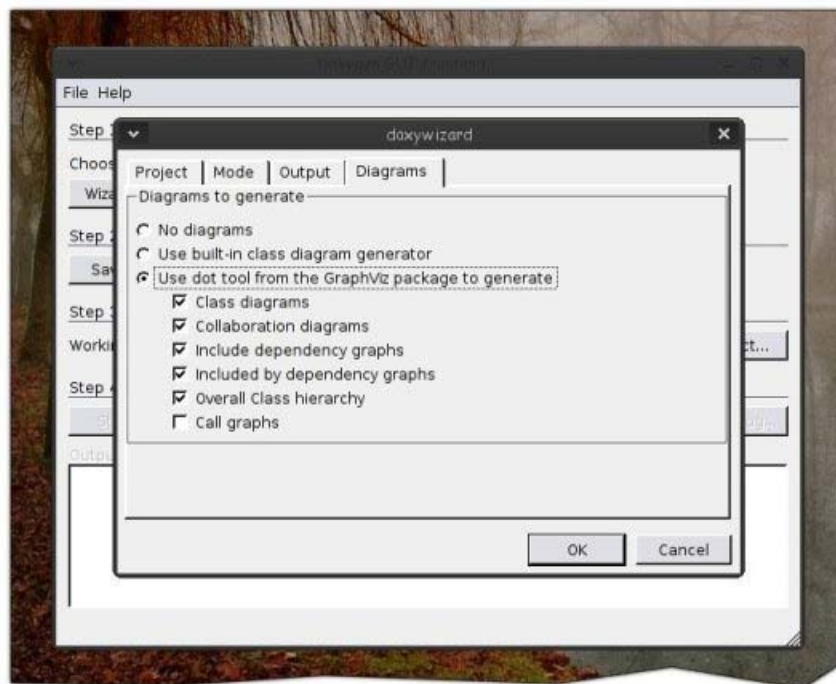


Ce troisième onglet permet le choix du type de sortie de la documentation. Plusieurs formats sont supportés: **HTML**, **LaTeX**, **Man pages** (*pages de manuels, Linux*), **Rich Text Format** (*fichier texte au format RTF*) et **XML**. Nous nous intéresserons surtout au format HTML dans ce tutoriel !

Le format HTML permet plusieurs type de présentations, les voici dans l'ordre :

1. Page sans zone de navigation, un peu à la manière de ce tutoriel si ont veut.
2. Page HTML avec une zone de navigation en arbre (*vue hiérarchique*) comme vous avez pu en voir dans les exemples de génération précédents.
3. Documentation au forma HTML compilé. C'est en fait le format des fichiers d'aide de Windows.
4. Cette option supplémentaire permet d'ajouter une zone de recherche, pratique pour de très grosses documentations mais, pour que ce module fonctionne,
5. il faut que PHP soit installé sur le serveur qui héberge, par exemple pour une version en ligne.

V-D. Onglet "Diagrams"

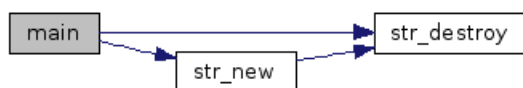


Comme il a déjà été mentionné, Doxygen permet également de créer des graphiques, notamment des graphiques de dépendances entre les différents fichiers donc les inclusions. C'est dans cet onglet que nous pouvons choisir les différents graphiques que nous voulons voir s'afficher dans la documentation !

Voici dans l'ordre, les graphiques proposés par Doxygen en utilisant le générateur **Graphviz** :

1. Créé un graphique montrant les relations directes et indirectes de chaque classes documentées. Cette option désactive automatiquement la génération de diagrammes intégrée à Doxygen (*l'option nommée "Use built-in classes diagram generator"*).
2. Créé un graphique pour les classes (et structures en ce qui concerne le C) montrant les relations avec les classes de bases ainsi que les relations avec d'autres structures/classes (ex: une structure A qui possède une variable membre du type d'une structure B).
3. Créé un graphique montrant les dépendances entre les différents fichiers documentés. Cette option ne fonctionne qu'avec la génération au format HTML et RTF !
4. Créé un graphique pour chaque fichier d'en-tête documenté montrant les fichiers documentés qu'inclut directement ou indirectement chaque fichier. Un sorte de graphique de dépendances entre chaque fichiers et les fichiers inclus.
5. Créé un graphique représentant la hiérarchie d'une classe, tout en incluant les dénominations textuelles. Supporté uniquement avec le format HTML !
6. Créé un graphique pour chaque fonction, montrant les fonctions que la fonction appelle directement ou indirectement. Cette option n'est utilisable que si vous cochez la case *"Include cross-referenced source code in the output"* dans le deuxième onglet !

Voici un exemple type d'un diagramme d'appel (*option "Call graphs"*). Ici il s'agit de la fonction *main* de l'exemple de code montré plus haut :



Important: pour utiliser GraphViz, il faut:

1. télécharger GraphViz pour Windows sur la [page suivante](#) et l'installer.
2. dans le doxywizard, cliquer sur le bouton **Expert**, sur l'onglet **Dot**, et ajouter le répertoire **bin** de GraphViz dans le champ DOT_PATH.
3. Il devient alors possible d'utiliser l'option "Use dot tool from GraphViz".

V-E. Valider les options et les enregistrer

Pour valider toutes ces options, il faut en premier lieu cliquer sur le bouton **Ok** de cette boîte de dialogue. Vous vous retrouverez

ensuite sur la fenêtre principale à partir de laquelle, vous pourrez enregistrer vos options dans l'étape 2 (*step 2*) par le bouton **Save** !

Mais avant de pouvoir lancer la génération par le bouton **Start**, il vous faut définir le répertoire courant dans la partie 3 (*step 3*), en générale je choisis le répertoire racine où se situe le code source.

VI. Configuration avancée

Je ne vais pas détailler ici toutes les options, ce serait bien trop long mais seulement certaines que j'estime être assez intéressantes pour peaufiner les réglages et la présentation générale de la documentation. Pour accéder aux options avancées de Doxygen, il faut passer par la fenêtre principale et cliquer sur le bouton

Expert qui se trouve juste après **Wizard** !



Chaque nom d'option dans les options avancées de Doxygen correspond en réalité aux options telles qu'elles sont écrites dans le fichier de configuration généré par le programme !

VI-A. Onglet "Project"

Une chose importante qu'il est bon à savoir, c'est que l'intitulé des différents paragraphes peut être généré en différentes langues, par défaut c'est en Anglais mais nous pouvons le mettre en Français, voyez pour ça, la liste nommée **OUTPUT_LANGUAGE**.

D'autres options intéressantes et utiles pour une utilisation plus ou moins basique de Doxygen:

- **USE_WINDOWS_ENCODING** : Permet de forcer l'encodage des caractères par celui utilisé par Windows si la case est cochée, sinon c'est un encodage de type Unix/Linux qui est utilisé.
- **BRIEF_MEMBER_DESC** : Permet d'afficher ou non la description courte dans la liste des fonctions/énumérations/structures/classes/etc... Liste qui se trouve en haut de page donc avant la partie détaillée de la documentation !
- **REPEAT_BRIEF** : Permet d'afficher ou non la description courte dans la partie détaillée de la documentation. Cette option et la précédente sont conjointement liées car si vous désactivez ces deux options, la description courte sera tout de même affichée dans la section détaillée de la documentation !
- **DETAILS_AT_TOP** : Permet d'afficher la description détaillée du fichier en haut de la page si cette option est cochée.

VI-B. Onglet "Build"

Pour peaufiner encore un peu plus nos réglages, les options suivantes peuvent être intéressantes:

- **EXTRACT_PRIVATE** (*C++ et autre langages orientés objet*) : Permet de faire afficher la documentation des fonctions/méthodes et autres membres ayant le qualificateur **private**.
- **EXTRACT_STATIC** : Permet de faire afficher la documentation des fonctions/structures/énumérations/etc... ayant le qualificateur **static**.
- **SORT_MEMBER_DOCS** : Si elle est cochée, cette option permet de trier les descriptions détaillées par ordre alphabétique sinon, l'ordre est celui de leur déclaration dans le code source.



Cocher l'option **EXTRACT_ALL** a le même effet qu'activer l'option "*All entities*" dans les options de l'assistant de configuration, sauf en ce qui concerne le style du tri !

VI-C. Onglet "HTML"

Vous désirez personnaliser la sortie de votre documentation ? C'est sur cet onglet que ça se passe alors ! Vous pouvez en effet ici, personnaliser l'en-tête et le pied de page de vos documentations en précisant leur chemin dans les champs **HTML_HEADER** et **HTML_FOOTER** respectivement. On peut également changer l'aspect général (*le design*) en précisant votre propre fichier CSS dans le champ **HTML_STYLESHEET** !

Dans le *chapitre 5*, j'avais également précisé que les générations se font par défaut dans un répertoire nommé **html**, c'est dans le champ **HTML_OUTPUT** que nous pouvons redéfinir le nom de ce répertoire, ainsi que le type de l'extension dans le champ **HTML_FILE_EXTENSION** !

L'option un peu plus bas **DISABLE_INDEX** permet de ne pas afficher l'index qu'on peut apercevoir en haut de chaque page HTML de la documentation !

VI-D. Onglet "Dot"

On peut également personnaliser un petit peu les graphiques générés, voici quelques options utiles:

- **UML_LOOK** : Permet de générer des diagrammes dans le style UML.
- **DOT_IMAGE_FORMAT** : Permet de choisir entre trois formats d'images différents soit **png**, **jpg** et **gif**.
- **DOT_PATH** : Cette option peut s'avérer utile lorsque vous avez un problème de détection du module de génération de graphiques et diagrammes. Cela peut arriver lorsque ce programme n'est pas référencé dans la variable *PATH*, il faut donc spécifier le chemin dans ce champ !
- **MAX_DOT_GRAPH_WIDTH** et **MAX_DOT_GRAPH_HEIGHT** : Permet de changer la taille maximale d'un graphe, respectivement la largeur et la hauteur.
- **MAX_DOT_GRAPH_DEPTH** : Permet de régler la profondeur maximum d'un graphe.
- **DOT_TRANSPARENT** : Permet de rendre le fond des images transparent, ça peut être très utile lorsqu'on personnalise l'aspect général de la documentation !

V. Bibliographie

- Ce document provient de [Franck Hecht](#), quelques éléments supplémentaires y ont été ajoutés avec l'autorisation de l'auteur.
- [François Lajeunesse-Robert](#).
- Site officiel de Doxygen: www.doxygen.org.