

Initiation à Doxygen pour C et C++

Par [Franck Hecht](#)  

Date de publication : 19 septembre 2007

Vous désirez créer de la documentation technique pour vos projets de développement ? Ce tutoriel va vous montrer la marche à suivre pour arriver à vos fins avec Doxygen !
Commentez cet article :

I - Introduction.....	3
II - Les balises standards.....	4
II-A - Les blocs de documentation.....	4
II-B - \file.....	4
II-C - \brief.....	4
II-D - \author.....	5
II-E - \version.....	5
II-F - \date.....	5
II-G - \struct.....	5
II-H - \enum.....	6
II-I - \union.....	6
II-J - \fn.....	6
II-K - \def.....	6
II-L - \param.....	6
II-M - \return.....	7
II-N - \bug.....	7
II-O - \deprecated.....	7
II-P - \class.....	7
II-Q - \namespace.....	8
III - Mise en place de la documentation.....	9
III-A - Informations d'en-tête.....	9
III-B - Documentation d'une fonction/méthode.....	9
III-C - Documentation d'une structure/union.....	9
III-D - Exemple complet en C.....	10
III-E - Exemple complet en C++.....	12
IV - Avant d'aller plus loin.....	14
V - Configuration de Doxygen avec Doxywizard.....	15
V-A - Onglet "Project".....	15
V-B - Onglet "Mode".....	16
V-C - Onglet "Output".....	16
V-D - Onglet "Diagrams".....	17
V-E - Valider les options et les enregistrer.....	18
VI - Configuration avancée.....	19
VI-A - Onglet "Project".....	19
VI-B - Onglet "Build".....	19
VI-C - Onglet "HTML".....	20
VI-D - Onglet "Dot".....	20
VII - Conclusion.....	21
VIII - Rappels des liens.....	22
IX - Remerciements.....	23

I - Introduction

A 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 détour vers des fonctions avancées pour générer des documentations de références comme par exemple celles de GTK+.

II - 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](http://frankh.developpez.com/tutoriels/outils/doxygen/)

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 ...
 */
```

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.
```

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 fourni en second argument.

Déclaration

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

Exemple

```
\struct Str_t
```

Exemple

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

II-H - \enum

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

Déclaration

```
\enum <name>
```

Exemple

```
\enum Str_err_e
```

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)
```

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 le chapitre).

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 le chapitre).

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
```


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. 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 ou une méthode de classes (C++), ce bloc ne change pas. Il faut noter qu'il doit y avoir une balise par argument. Tous les paramètres seront alors dans un même paragraphe **Paramètres** !

Exemple pour une fonction en C

```
/**
 * \fn static Str_t * str_new (const char * sz)
 * \brief Fonction de création d'une nouvelle instance d'un objet Str_t.
 *
 * \param sz 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, il faut fournir la déclaration complète de la fonction ou de la méthode !



Il ne faut surtout pas oublier de bien fournir la balise 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 . 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

```
Énumération:  
    STR_NO_ERR      Pas d'erreur.  
    STR_EMPTY_ERR   Erreur: Objet vide ou non initialisé.  
    NB_STR_ERR      Nombre total de constantes d'erreur.
```

III-D - Exemple complet en C

Voici un exemple de génération de la documentation de cet exemple : **Exemple**

```
/**  
 * \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  
 * connaît 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 self Adresse de ll'objet Str_t à détruire.
 * \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 sz 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 chaine de caracteres !");

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

        err = str_destroy (& my_str);

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

    return EXIT_SUCCESS;
}

```

III-E - Exemple complet en C++

Voici un exemple de génération de la documentation de cet exemple : **Exemple**

```

#ifndef CPLAYER_H_
#define CPLAYER_H_

/*!
 * \file CPlayer.h
 * \brief Lecteur de musique de base
 * \author hiko-seijuro
 * \version 0.1
 */
#include <string>
#include <list>

/*! \namespace player
 *
 * espace de nommage regroupant les outils composants
 * un lecteur audio
 */
namespace player
{
    /*! \class CPlayer
     * \brief classe representant le lecteur
     *
     * La classe gere la lecture d'une liste de morceaux
     */
    class CPlayer
    {
    private:
        std::list<string> m_listSongs; /*!< Liste des morceaux*/
        std::list<string>::iterator m_currentSong; /*!< Morceau courant */

    public:
        /*!
         * \brief Constructeur
         *
         * Constructeur de la classe CPlayer
         *
         * \param listSongs : liste initial des morceaux
         */
        CPlayer(std::list<string> listSongs);

```

```
/*!
 * \brief Destructeur
 *
 * Destructeur de la classe CPlayer
 */
virtual ~CPlayer();

public:
/*!
 * \brief Ajout d'un morceau
 *
 * Methode qui permet d'ajouter un morceau a liste de
 * lecture
 *
 * \param strSong : le morceau a ajouter
 * \return true si morceau deja present dans la liste,
 * false sinon
 */
bool add(std::string strSong);

/*!
 * \brief Morceau suivant
 *
 * Passage au morceau suivant
 */
void next();

/*!
 * \brief Morceau precedent
 *
 * Passage au morceau precedent
 */
void previous();

/*!
 * \brief Lecture
 *
 * Lance la lecture de la liste
 */
void play();

/*!
 * \brief Arret
 *
 * Arrete la lecture
 */
void stop();
};


#endif
```

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 inclus le tout*) alors installer plusieurs programmes :

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

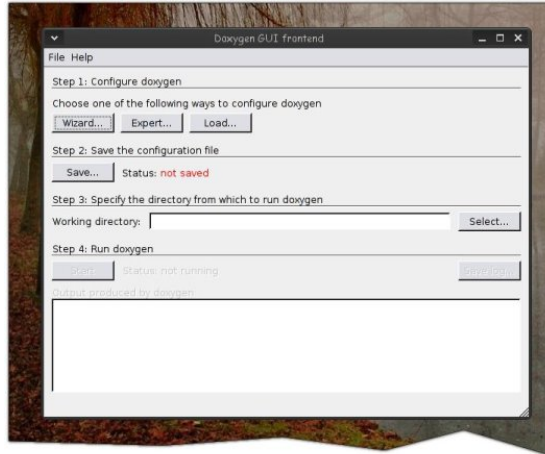
Dont  **graphviz** permet la génération de graphiques (*entre autres, des graphiques de dépendances... oui oui, il fait également ça ce merveilleux programme !*) et **doxygen-gui** est l'interface graphique (*doxywizard*) !

V - Configuration de Doxygen avec Doxywizard

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

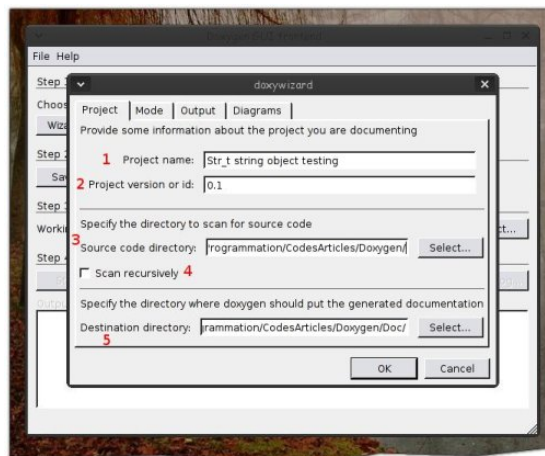
```
franhec@lucie:~$ doxywizard
```

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



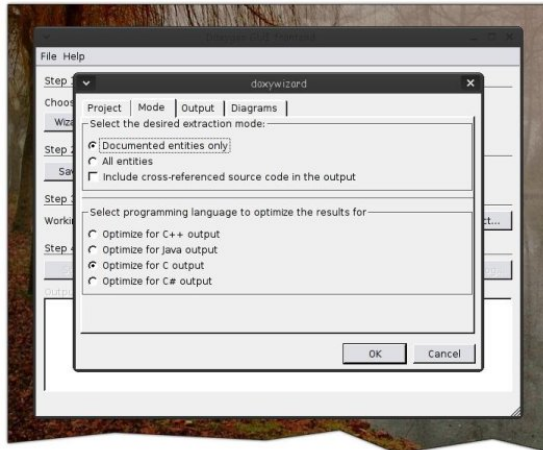
Nous allons maintenant voir comment configurer un projet. Pour commencer, nous passons par l'assistant donc le bouton **Wizard...**

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"



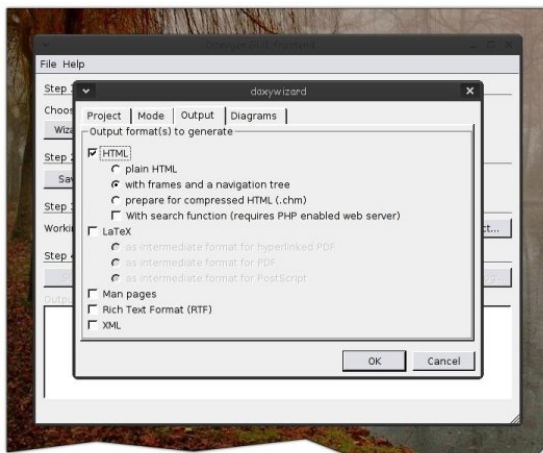
Dans ce second onglet d'options, dans la première série nous pouvons déterminer le détail du contenu (*plus ou moins exactement*). Par défaut l'option est sur **Documented entities only**, ceci permettra simplement la génération de la documentation des parties documentées, les autres ne seront pas référencées. 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 parties documentées 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"



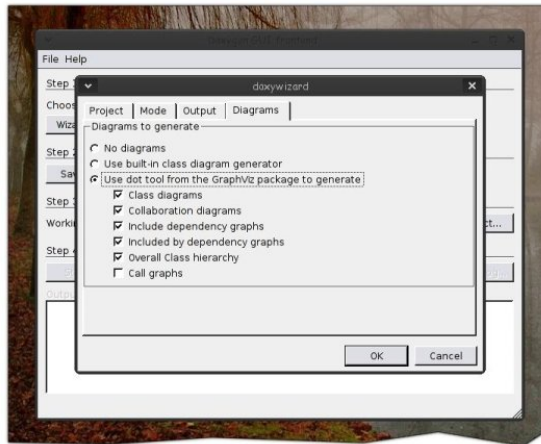
Ce troisième onglet permet le choix du type de sortie de la documentation. Plusieurs formats sont supportés: **HTML**, **LaTeX**, **Man pages**(*pages de manuels, Linux*), **Rich Text Format**(*fichier texte au format RTF*) et **XML**. Nous nous intéresserons surtout au format HTML dans ce tutoriel !

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 format 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, il faut que PHP soit installé sur le serveur qui héberge, par exemple pour une version en ligne.

V-D - Onglet "Diagrams"

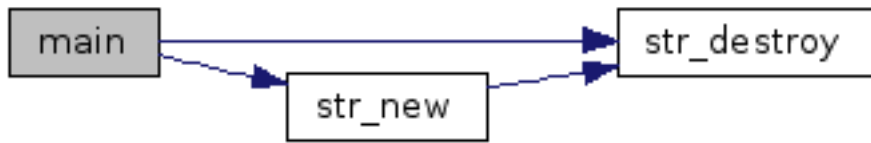


Comme il a déjà été mentionné, Doxygen permet également de créer des graphiques, notamment des graphiques de dépendances entre les différents fichiers donc les inclusions. C'est dans cet onglet que nous pouvons choisir les différents graphiques que nous voulons voir s'afficher dans la documentation !

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 :



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.

Les options telles que présentées dans les captures d'écrans ont pour résultat la sortie suivante pour l'exemple de code en **C** et à quelques détails près pour l'exemple en **C++**

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 possible de cette page sont :

- **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.

Voilà quelques options utiles pour une utilisation plus ou moins basique de Doxygen. L'activation de ces options a pour résultat la génération suivante pour les codes d'exemples:

- **C**
- **C++**

On peut déjà voir des changements notamment le titre en gras des paragraphes qui est désormais en Français !

VI-B - Onglet "Build"

Dans cette section, pour peaufiner encore un peu plus nos réglages de génération, seule quelques options nous intéresserons à savoir:

- **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.

Voilà le résultat de la génération après avoir coché les options **EXTRACT_STATIC** et **EXTRACT_PRIVATE** respectivement pour le C et le C++ et décoché l'option **SORT_MEMBER_DOCS** :

- **C**
- **C++**



Cocher l'option **EXTRACT_ALL** à 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, ca peut être très utile lorsqu'on personnalise l'aspect général de la documentation !

VII - Conclusion

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 de GTK+ ou Glib par exemple. Voici une documentation que j'ai fait pour une de mes bibliothèques personnelles: **C_Str v2.1.0 - Documentation v1.2** ... ça mérite de s'attarder un peu sur ce programme non ?

VIII - Rappels des liens

Je vous rappelle ici les liens cités dans ce tutoriel et quelques autres encore:

-  **Page d'accueil du site officiel**
-  **La FAQ**
-  **Page principale du manuel en ligne**
-  **Créer des listes dans les documentations**
-  **Liste de toutes les balises**
-  **Description de toutes les options**

IX - Remerciements

Merci à **hiko-seijuro** pour le code d'exemple en version C++ et à **Aspic** pour la relecture et correction de ce tutoriel !