Niveau 3

Tutoriel
S2 - Gestion de projet
S2.2 - Documentation
Outils de génération de documentation logicielle

Documenter son code avec doxygen

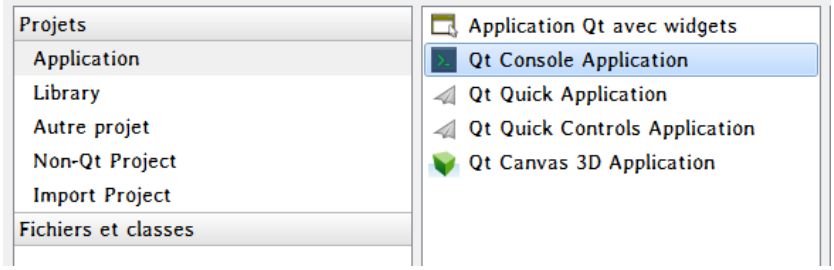
TABLE DES MATIÈRES

CERTAINS DROITS RESERVES

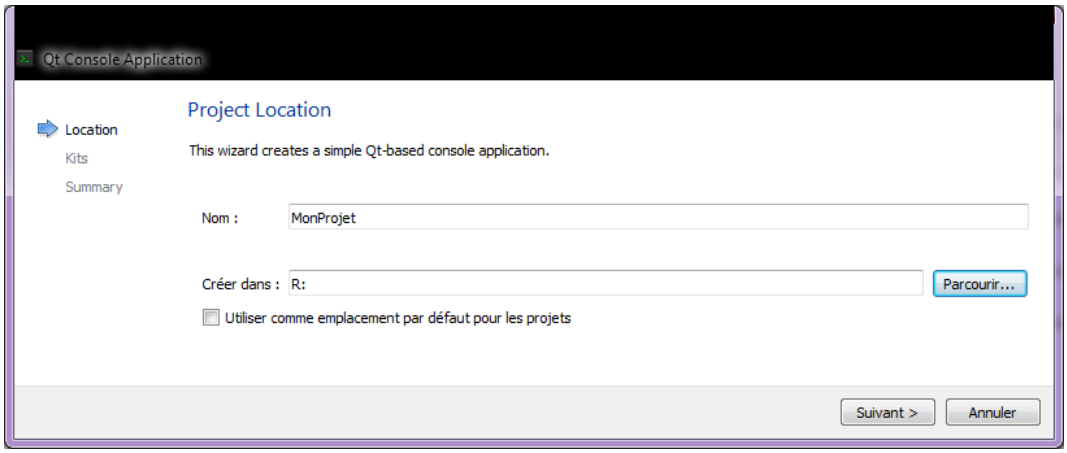
1 - PRÉPARATION D'UN PROJET C++ AVEC QT

Pour créer un nouveau projet au moyen de QtCreator, il suffit, après l'avoir lancé, de sélectionner dans la page `Accueil` l'option 

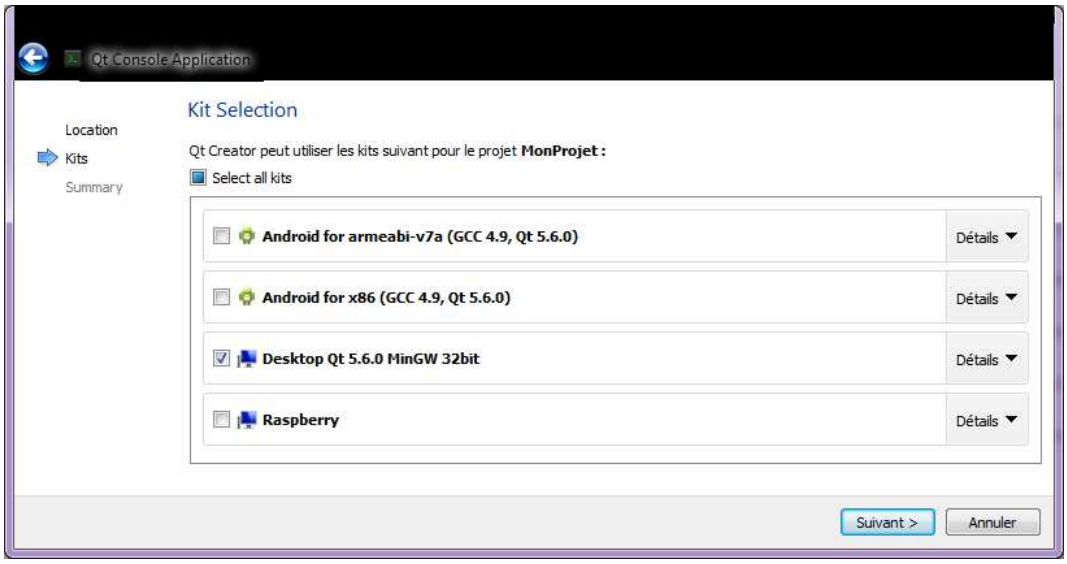
Après avoir choisi le type d'application que vous souhaitez réaliser :



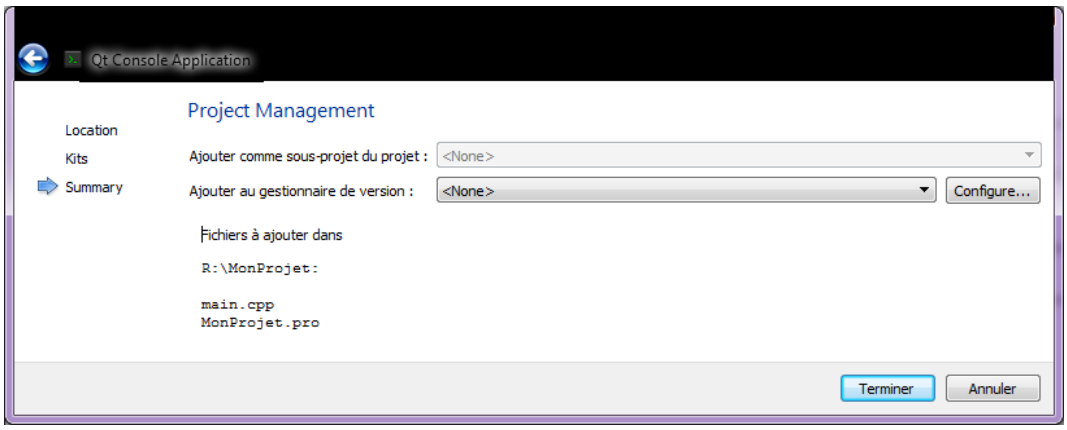
Il vous est proposé dans une nouvelle fenêtre de renseigner le nom (`MonProjet`) et l'emplacement du répertoire de votre projet (ici la racine d'un disque nommé `R:`) :



Un clic sur suivant vous permet d'accéder au choix du kit de développement à utiliser. Généralement, il s'agit du `Desktop Qt MinGW` mais ce choix peut être multiple :



La prochaine fenêtre vous proposera d'intégrer votre projet dans un autre et de sélectionner le système de reposery à utiliser (à laisser vide pour le moment, ça ne servira qu'en projet de 2nde année) :



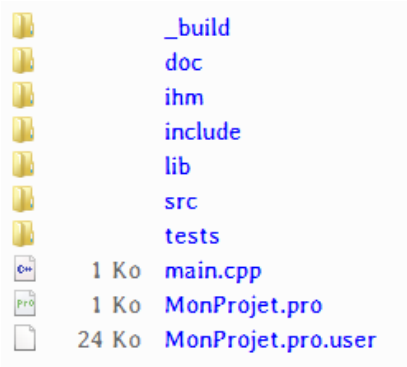
La base de votre projet est prête et se trouve dans le répertoire `R:/MonProjet`. Automatiquement, le fichier du projet (`MonProjet.pro`), son fichier de configuration lié à votre session et ordinateur (`MonProjet.pro.user`) et le `main.cpp` sont créés.

> Attention de ne pas effacer le fichier `.pro` ; votre projet s'en trouverait inutilisable et il faudra le refaire ! Par contre, le fichier `.pro.user`, étant lié à la configuration de votre ordinateur, il devra être supprimé dès que vous changez de machine. Ne vous inquiétez pas, QtCreator en générera un nouveau adapté à la nouvelle machine.

Avant de commencer la programmation, il convient d'arranger correctement la structure de ce répertoire afin de ranger les fichiers comme il faut.

Vous devrez créer de nouveaux directories pour les fichiers sources (`src`), exécutables et aides de compilation (`_build`), librairies (`lib`), en-têtes (`include`), les programmes de test (`tests`) et documentation (`doc`) de votre projet.

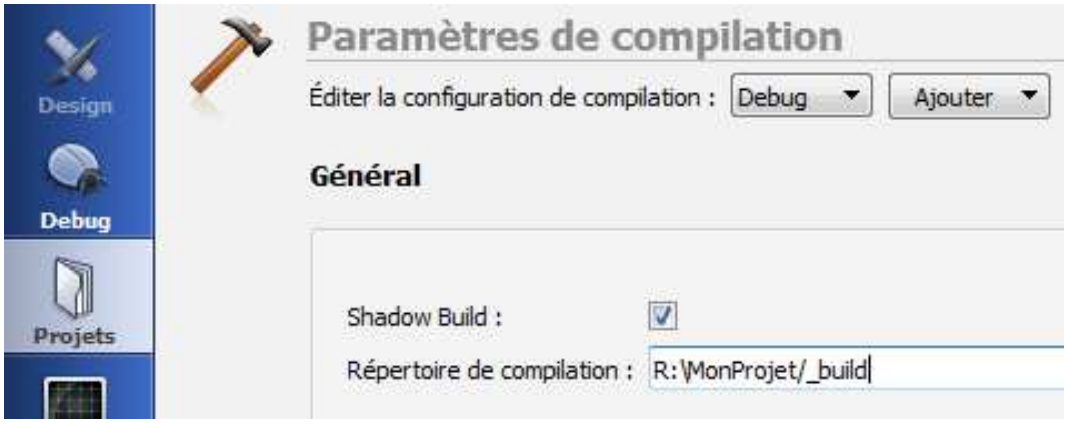
Pour cela, dans l'explorateur de votre système d'exploitation, créez une arborescence telle que montrée :



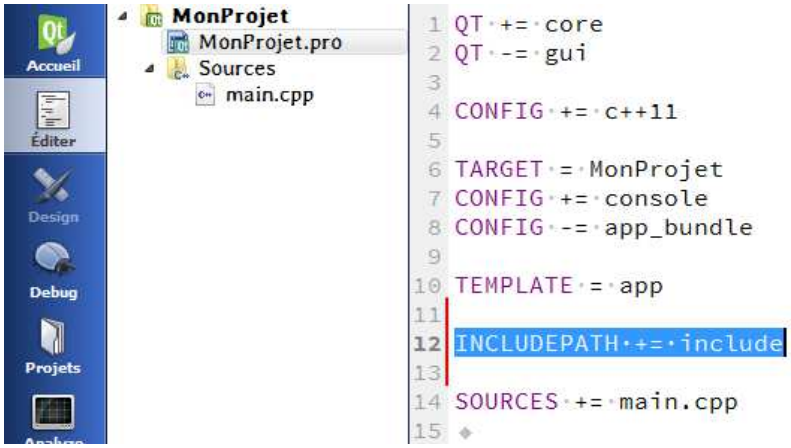
Votre répertoire de projet est correctement organisé !

Vous allez à présent devoir configurer les options du projet de Qt pour que cette architecture soit correctement utilisée.

Dans l'onglet `Projet` de QtCreator, modifiez le répertoire de compilation du `Shadow Build` (qui doit être coché par défaut) et fixez-le sur `R:/MonProjet/_build` :



Sélectionnez l'onglet `Éditer` et double cliquez sur `MonProjet.pro` afin de l'éditer. Ajoutez la ligne `INCLUDEPATH` comme montrée :



Votre projet est prêt. Vous pouvez essayer de compiler pour le vérifier !

2 - CONFIGURER LA DOCUMENTATION

Un programme doit être maintenable facilement, même lorsque les membres de l'équipe changent.

Il est nécessaire de bien documenter techniquement les diverses parties d'un projet pour faciliter le débogage et le suivi des évolutions.

Cette étape peut-être rendue moins fastidieuse en utilisant des outils de documentation automatique qui permettent de générer une documentation au plus près du code. Ils se servent des commentaires placés dans le code lui-même pour fournir une documentation complète, et cela de façon très facile pour le programmeur !

nous allons utiliser dOxygen (https://www.doxygen.nl/download.html).

2.1 - À PARTIR DE LA CONFIGURATION PAR DÉFAUT

Vous pouvez générer le fichier de configuration `Doxyfile.cfg` directement par dOxyGen, dans une console, en entrant la commande :

```
doxygen -g Doxyfile.cfg
```

Attention toutefois, par cette opération, les paramétrages fournis seront ceux par défaut de dOxyGen : vous aurez de grosses modifications à faire !

Commencez par ajouter un logo à votre projet, qui sera intégré à la documentation. Préparez une image nommé `Logo.png` avec votre éditeur graphique préféré, et placez-la dans le directory `R:/MonProjet/doc`. Attention à la taille de cette image : sa taille ne devrait pas dépasser 55 × 200 pixels, mais rien n'est imposé.

Lancez ensuite la console de votre système d'exploitation, déplacez-vous dans le répertoire `R:/MonProjet/doc` et générez le `Doxyfile.cfg` :

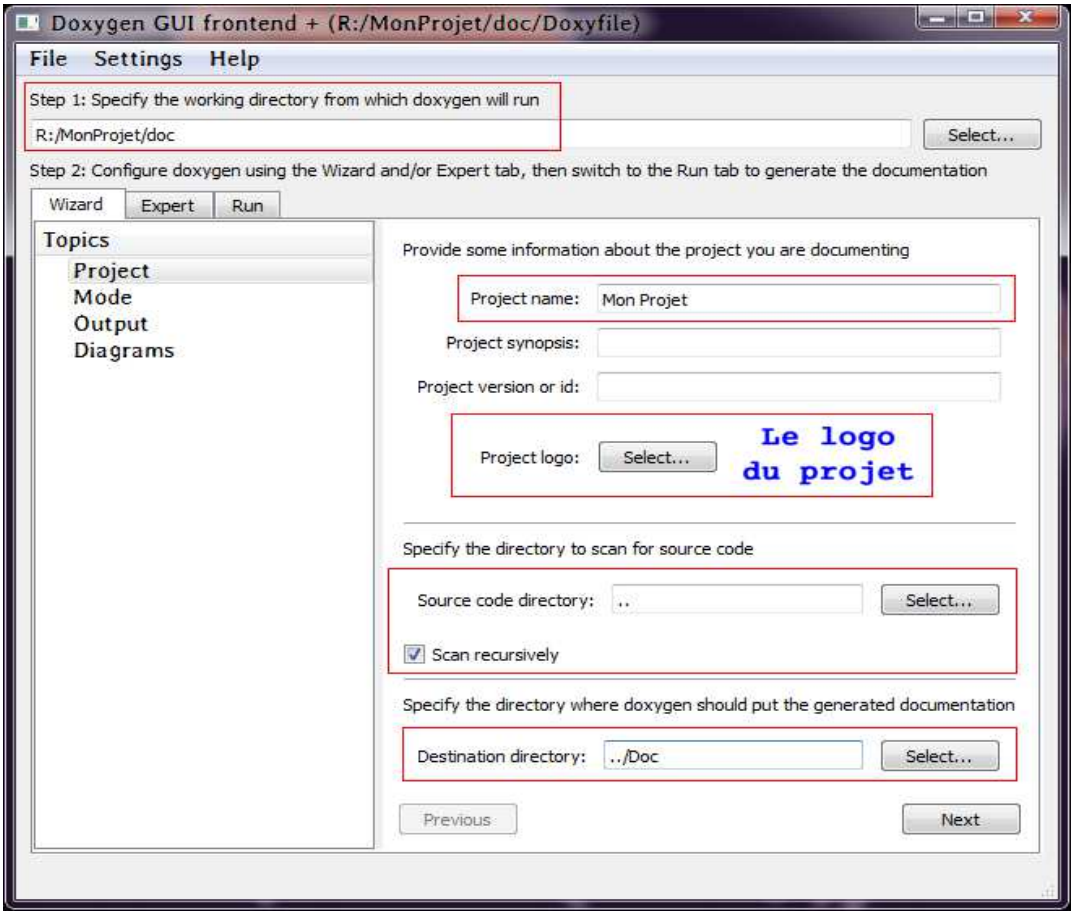
```
R:
cd R:/MonProjet/doc
doxygen -g Doxyfile.cfg
```

2.1.1 - MODE WIZARD

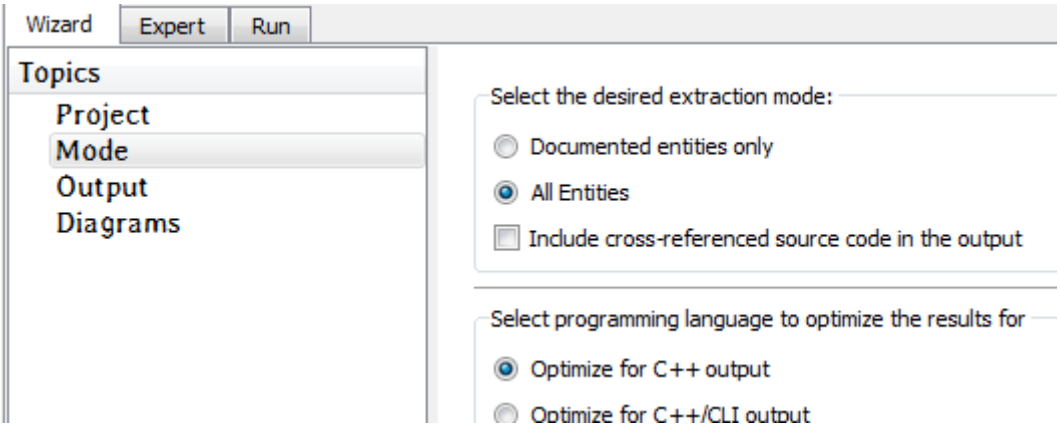
Une fois le fichier obtenu, lancez l'utilitaire dOxyWizard et chargez le `Doxyfile.cfg` désiré par le menu `File/Open…`.

Dans le champ de saisie `Step 1`, renseignez le chemin d'exécution `R:/MonProjet/doc`.

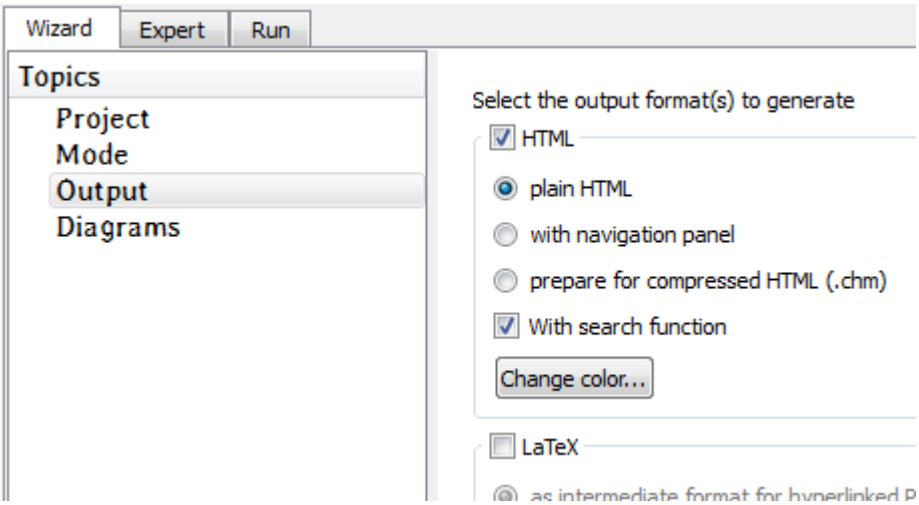
Dans l'onglet `Wizard`, option `Project`, modifiez les champs `Project name` (`Mon Projet`), `Source code directory` (`..`) et `Destination directory` (`../doc`). Cochez l'option `Scan recursively` et sélectionnez, comme `Project logo`, l'image `Logo.png` que vous avez précédemment réalisée.



L'appui sur `Next` vous fait passer à l'option `Mode`. Sélectionnez le mode d'extraction `All entities` à la place de `Documented entities only`. L'optimisation pour le C++ est par défaut, pas de modification à apporter donc !



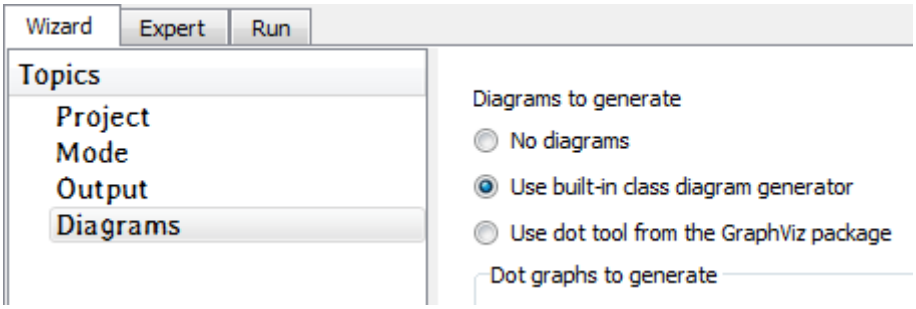
L'option suivante, `Output`, vous permet de choisir le format de la documentation souhaitée. Par défaut, `HTML` et `Latex` sont cochées. Décochez le format `Latex` qui ne nous est pas utile et éventuellement l'option de recherche de la partie `HTML` si vous n'en voulez pas.



Notez que vous pouvez configurer la couleur générale de votre site documentaire en cliquant sur le bouton `Change color...`.

L'option `Diagrams` vous propose de compléter votre documentation avec des diagrammes de classes, de collaboration...

Par défaut, les diagrammes de classes sont créés au moyen du générateur interne de dOxyGen. À moins d'avoir installé le générateur GraphViz sur votre poste, laissez l'option `Use built-in class diagram generator` cochée :



Voir le chapitre 5.1 - Amélioration des diagrammes pour installer et utiliser GraphViz.

La configuration de base est terminée, nous allons maintenant passer à la configuration avancée.

2.1.2 - MODE EXPERT

Le mode expert vous propose une configuration plus approfondie de dOxyGen. Tous les paramétrages réalisés durant le mode `Wizard` sont, bien sûr, maintenus.

Attention, dans cette partie, seules les paramètres à vérifier, valider ou modifier vous seront présentés topic par topic dans les sous chapitres appropriés.

Une fois les vérifications et modifications apportées, sauvegardez le fichier `Doxyfile.cfg` par le menu `File/Save`. Attention à ne pas oublier l'extension !

Votre fichier `Doxyfile.cfg` est utilisable !

Vérifiez que les paramètres indiqués ont bien les valeurs suivantes :

2.1.2.1 - PROJECT

Paramètre	*Valeur*
PROJECT_NAME	Mon Projet
PROJECT_LOGO	Logo.png (sans le chemin)
OUTPUT_DIRECTORY	../doc
CREATE_SUBDIRS	Non coché
OUTPUT_LANGUAGE	French
TAB_SIZE	4
BUILTIN_STL_SUPPORT	Coché
CPP_CLI_SUPPORT	Décoché

2.1.2.2 - BUILD

Paramètre	*Valeur*
EXTRACT_ALL	Coché
EXTRACT_PRIVATE	Coché

2.1.2.3 - INPUT

Paramètre	*Valeur*
INPUT	.. description.txt
RECURSIVE	Coché

2.1.2.4 - SOURCE BROWSER

Paramètre	*Valeur*
INLINE_SOURCE	Coché

2.2 - À PARTIR D'UNE CONFIGURATION EXISTANTE

Depuis un autre projet, ayant la même structure arborescente, vous pouvez récupérer le fichier de configuration de la génération de la documentation : `Doxyfile.cfg`.

Copiez ce fichier dans le répertoire `R:/MonProjet/doc` et accompagnez-le d'un nouveau fichier `Logo.png` lié à votre nouveau projet. Ces deux fichiers doivent être dans le même directory !

Lancez le dOxyWizard et ouvrez ce `Doxyfile.cfg`, afin d'effectuer de légères modifications : changez le nom de votre projet et éventuellement le numéro de version, s'il est renseigné.

Vérifiez ensuite rapidement que les options des modes `Wizard` (cf. 2.1.1 - Mode Wizard) et `Expert` (cf. 2.1.2 - Mode Expert) sont valides.

Sauvegardez. Votre `Doxyfile.cfg` est prêt.

> Notez que vous pouvez tout aussi bien éditer le `Doxyfile.cfg` avec n'importe quel éditeur de texte simple, tel que le notepad, notepad++ ou encore SciTE. Dans ce cas, il vous faudra rechercher le paramètre `PROJECT_NAME` et modifier sa valeur.

3 - GÉNÉRATION DE LA DOCUMENTATION

3.1 - EN LIGNE DE COMMANDE

Lancez une console et placez-vous dans le directory `R:/MonProjet/doc`.

Exécutez la commande `doxygen Doxyfile.cfg`. Patientez 5 secondes... votre documentation est prête !

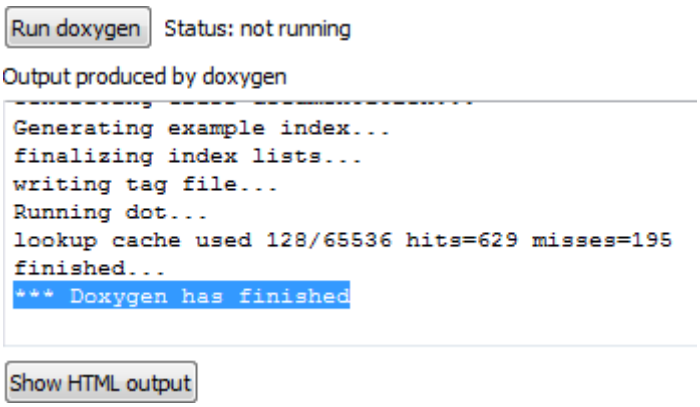
```
cd R:/MonProjet/doc
doxygen Doxyfile.cfg
```

Vous pouvez l'afficher dans n'importe quel navigateur en ouvrant la page `R:/MonProjet/doc/html/index.html`.

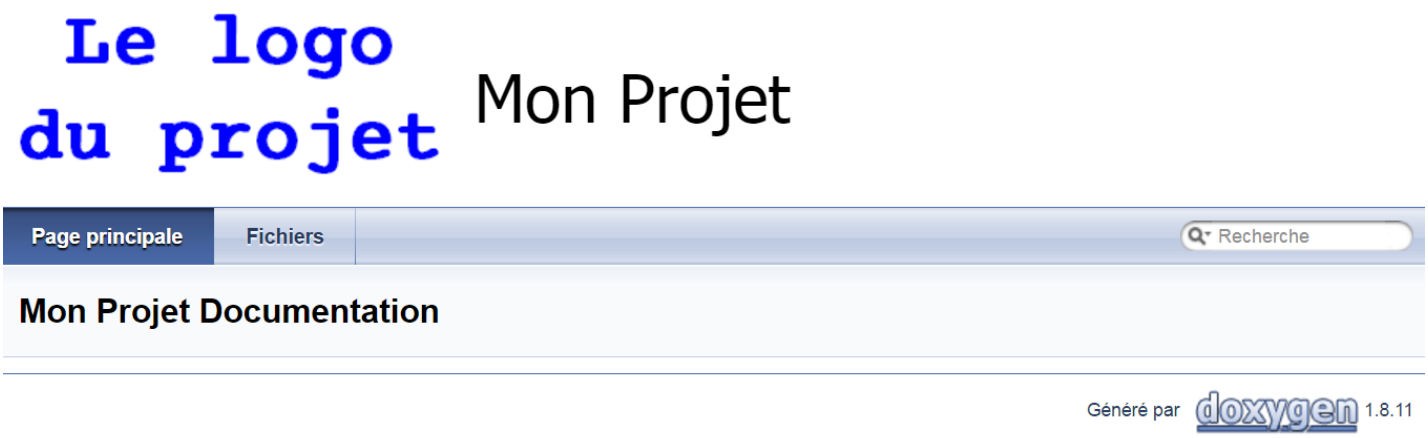
3.2 - AVEC LE DOXYWIZARD

Quel que soit votre système d'exploitation, lancez dOxyWizard et chargez le `Doxyfile.cfg` (menu `File/Open…`).

Allez dans l'onglet `Run`, et cliquez sur le bouton [Run doxygen]. Patientez...



Lorsque vous voyez le message `*** Doxygen has finished`, vous pouvez visualiser votre documentation par un clic sur le bouton Show HTML output :



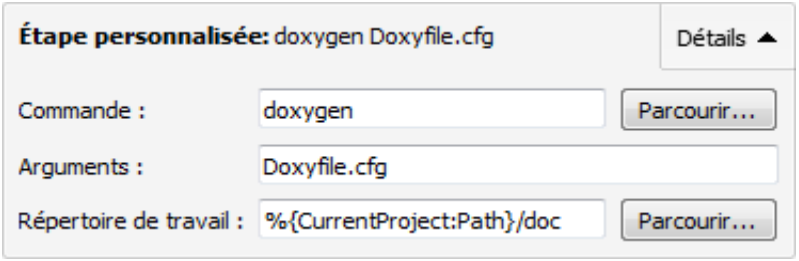
3.3 - QTCREATOR

Nous allons maintenant ajouter la génération de la documentation dans notre projet avec QtCreator.

3.3.1 - DURANT LA PHASE DE COMPILATION

Allez dans l'onglet `Projet` et ajoutez une étape de compilation personnalisée en cliquant sur le bouton Ajouter l'étape Compiler et en choisissant `Étape personnalisée` dans le menu contextuel.

Complétez cette étape de compilation avec les valeurs suivantes :



Dès à présent, lors d'une compilation de votre projet, la documentation sera automatiquement mise à jour !

Cette configuration sera à refaire pour chaque nouveau projet...

Attention, pour de gros projets, cette étape supplémentaire prendra du temps et ralentira la phase de compilation globale.

Il ne vous reste plus qu'à coder votre application et surtout à commenter les divers fichiers et méthodes.

Notez qu'en cas d'erreur de génération de la documentation, notamment parce que le fichier `Doxyfile.cfg` n'est pas trouvé, il vous suffit de quitter QtCreator et de supprimer le fichier `.pro.user` à la racine de votre directory de projet. Relancez QtCreator. Il vous sera demandé de configurer le projet. Acceptez et ajoutez de nouveau l'étape personnalisée de compilation dans l'onglet `Projet`. N'oubliez pas de modifier aussi le chemin du `Shadow Build`, comme indiqué à la page 4.

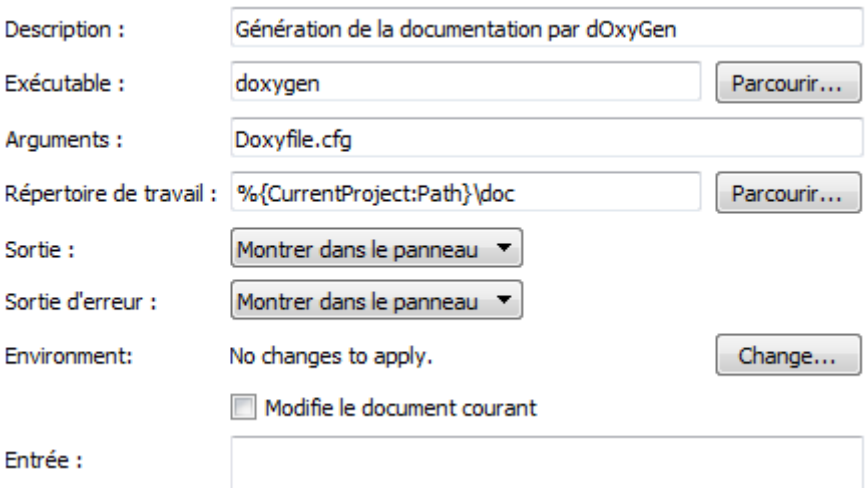
3.3.2 - EN AJOUTANT UN OUTIL EXTERNE

Pour ne pas surcharger la phase de compilation, vous pouvez créer une nouvelle entrée dans le menu `Outils/Externes/Textes`. L'avantage est, qu'une fois ce menu créé, il reste disponible pour les projets futurs !

Allez dans le menu `Outils/Externes/Configure...`.

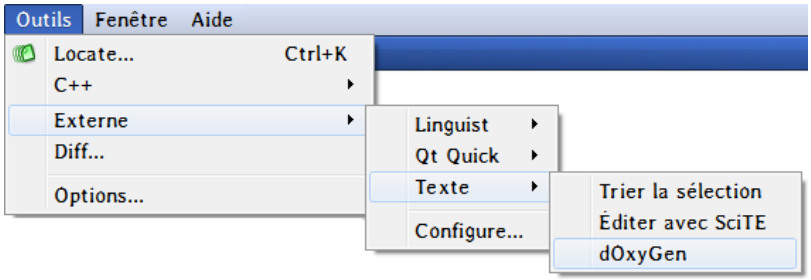
Choisissez la catégorie `Textes`, et cliquez sur le bouton `Ajouter` pour obtenir le menu contextuel, dans lequel vous choisissez l'option `Ajouter un outil`, que vous nommerez `dOxyGen`.

Renseignez les divers champs comme montré ci-dessous :



Validez.

Vous aurez à présent une entrée `Outils/Externes/Textes/dOxyGen` dans la barre de menu.



Lancez cette commande lorsque vous souhaitez générer manuellement la documentation.

La génération et mise à jour de la documentation étant manuelle, n'oubliez pas de la faire !

4 - COMMENTER UN PROJET

4.1 - DIRECTEMENT EN COLLANT LE COMMENTAIRE

Le format des commentaires à appliquer à chaque structure de votre projet, que ce soit les fichiers, les méthodes ou autres, est bien défini dans les règles de programmation de la section.

Vous en retrouverez certaines dans le document global des normes de programmation en vigueur dans la section SN-IR, ou bien, plus précisément, dans ce document-ci au chapitre 4.3 - Format des commentaires.

Il ne vous sera pas difficile de faire jouer vos talents avec les touche `Ctrl-C` et `Ctrl-V`... Je vous fait confiance pour ça !

4.2 - AVEC UN SNIPPET

Un snippet est l'association d'un alias, généralement assez court, et d'une combinaison de touches que votre EDI propose afin de vous éviter de taper la totalité d'une structure algorithmique qui est utilisée fréquemment.

4.2.1 - SOUS QTCREATOR

Par exemple, avec QtCreator, écrivez simplement le mot clef `for`, puis pressez les touches `Ctrl` et `Espace` simultanément. QtCreator va remplacer votre `for` avec la structure correspondante complète !

S'il existe plusieurs possibilités, il vous sera demandé de choisir parmi toutes dans un menu contextuel.

4.2.1.1 - PRÉPARER LE SNIPPET

Si la plupart des structures communes sont déjà implantées dans les snippets de QtCreator, d'autres, plus adaptés à votre usage, peuvent être ajoutés.

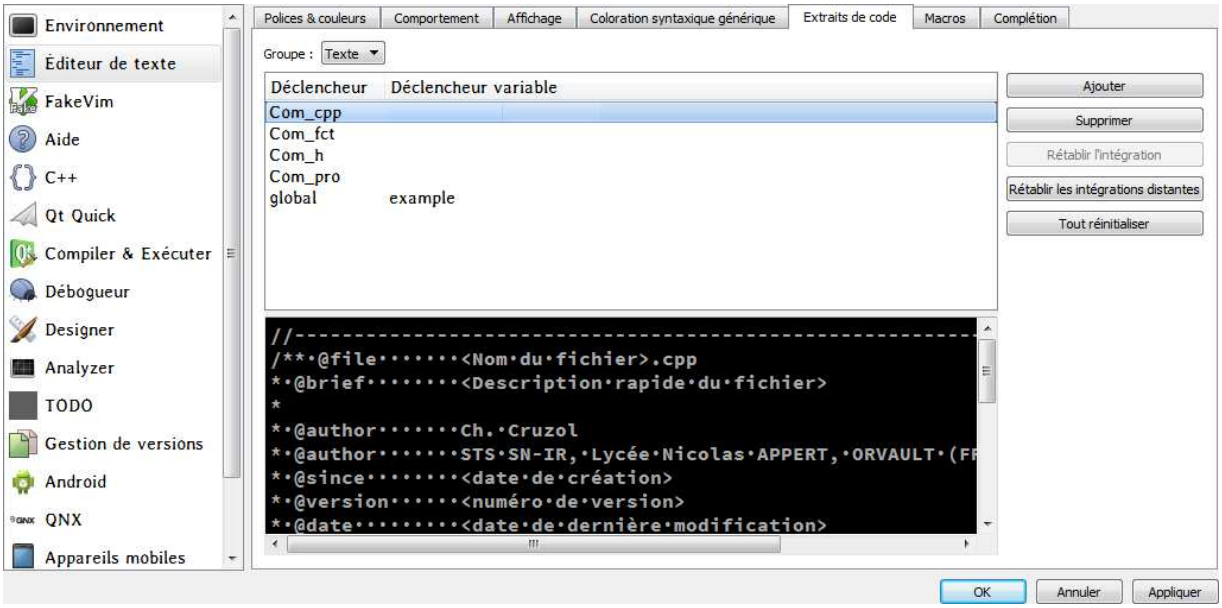
Nous allons créer un snippet pour nous éviter de taper la totalité du commentaire d'en-tête des fichiers cpp. La démarche est identique pour les snippets des autres commentaires.

Ouvrez la fenêtre de configuration de QtCreator au travers de le menu `Outils/Options…`.

Dans le module `Éditeur de texte`, sélectionnez l'onglet `Extrait de code`.

Cliquez sur `Ajouter` et renseignez le nom de l'alias à utiliser, par exemple `Com_cpp`. Validez par `Entrée`.

En faisant bien attention d'avoir sélectionné ce nouveau snippet, collez, dans la zone d'édition, le gabarit du commentaire d'en-tête des fichiers cpp.



Il n'est pas besoin de valider pour le moment.

Il vous suffit d'ajouter tout les snippets de commentaire que vous voulez, selon la même démarche.

J'en ai créé trois de plus dans cet exemple : `Com_fct`, `Com_h` et `Com_pro`.

Lorsque tous vos snippets sont ajoutés, quittez la fenêtre de configuration par un clic sur le bouton `OK`.

Vos snippets sont utilisables dès à présent.

4.2.1.2 - UTILISER LE SNIPPET

Lorsque vous êtes en train de programmer, il vous suffira de saisir l'alias du snippet (par exemple `com_cpp`) à l'endroit désiré et de presser simultanément les touches `Ctrl` et `Espace`.

C'est tout !

4.3 - FORMAT DES COMMENTAIRES

Il existe plusieurs formats utilisables par dOxyGen, selon le langage de programmation.

Dans la section, nous avons choisi d'utiliser une implémentation de commentaires compatible avec Javadoc, un autre générateur automatique très utilisé dans le monde informatique.

Que vous programmiez en C++ ou en Java, vous pourrez ainsi utiliser les mêmes commentaires de documentation !

4.3.1 - LE DESCRIPTIF DU PROJET

Le descriptif du projet peut être placé dans un fichier séparé de votre projet, par exemple dans `R:/MonProjet/doc/description.txt`, ou être inclus dans le `main.cpp`.

Dans le cas où vous le placez dans un fichier indépendant, pensez à modifier votre `Doxyfile.cfg` de façon à ce que dOxyGen le retrouve. Pour cela, allez dans le mode `Expert` du dOxyWizard, et ajoutez ce fichier dans le topic `Input` à l'option `INPUT` :

Vous pouvez aussi modifier directement le `Doxyfile.cfg` dans n'importe quel éditeur de texte. Recherchez l'entrée `INPUT` et adaptez-la :

```
INPUT = .. \
        description.txt
```

Cette solution vous permet de réutiliser plus facilement ce fichier d'un projet à l'autre, à l'instar du `Doxyfile.cfg`. Il vous suffira seulement de les placer tous les deux dans le même directory, et le tour sera joué.

Voici maintenant la structure générale de la page de garde de votre documentation :

```
/** @mainpage Test de commentaires
 *
 * @section Introduction Introduction
 *
 * Le but de ce projet est de tester les diverses formes de commentaires proposées par
 * dOxygen afin de
 * définir une structure stable et fonctionnelle pour la plupart des projets de SN-!R
 * du lycée Nicolas Appert.
 *
 * @section Utilisation Utilisation
 * @subsection Librairie Récupérer les binaires
 *
 *     $ commande style git clone https://github.com/ etc.
 *     $ Pour une ligne encadrée comme ça, placez 5 espaces entre l'étoile et le $ et
 *       une lignevierge avant !!!
 *     $ Deux lignes successives seront encadrées par le même cadre !
 *
 * @subsection Sources Récupérer les codes sources
 * @subsection Compilation Compilation des sources
 * @subsection Installation Installation
 * Il n'est pas besoin d'installer ce projet : ce n'est pas une librairie !
 *
 * @subsection MEO Mise en œuvre
 * @subsection Exécution Exécution
 *      - Au démarrage
 *      - En console en mode administrateur (obligatoire pour avoir les droits de
*         commander l'USB).
 *
 * @section Exigences Exigences
 * @subsection Logiciels Logiciels
 *      - Librairie TrucMuch
 * @subsection Matériels Matériels
 *      - Raspberry Pi (toute version)
 *
 * @section Remarques Remarques
 * @section Liens Liens
 *       -# [Documentation de dOxygen](http://www.stack.nl/~dimitri/doxygen/manual)
 *       -# [QtCreator](https://www.qt.io/download/)
 *       -# [Geany](http://www.geany.org/Download/Releases)
 *
```

```
 * @section Auteur Auteur
 * Ch. Cruzol, STS SN-IR, 2016-05-06.
 */
```

Il vous est possible d'ajouter, de modifier ou de supprimer n'importe quelle section afin de personnaliser votre page de garde.

4.3.2 - LES FICHIERS .CPP

Chaque fichier cpp devra être documenté en tout début, avant les lignes de code.

```
//-------------------------------------------------------------------------
/** @file        main.cpp
 *
 *  @brief       Programme de mise au point et de test des diverses fonctionnalités.
 *
 *  @author      Ch. Cruzol
 *  @author      STS SN-IR, Lycée Nicolas APPERT, ORVAULT (FRANCE)
 *  @since       2015-11-06
 *  @version     1.0
 *  @date        2016-05-06
 *
 *  Description détaillée du fichier
 *
 *  Fabrication   MonProjet.pro
 *
 *  @todo        Rien
 *
 *  @bug         Aucun
 */
//-------------------------------------------------------------------------
```

4.3.3 - LES FICHIERS .H

Comme ce sont des fichiers en-tête liés aux fichiers .cpp, leurs commentaires ne sont pas développés pour ne pas être redondants avec ceux des .cpp.

Ce commentaire est à mettre sur la première ligne du fichier.

```
// nom.h      num.version      date de création      Développeur
```

4.3.4 - LES CONSTANTES SYMBOLIQUES

```
/** @def NBRE_MAXIMAL
 *  @brief Description rapide de la constante symbolique NBRE_MAXIMAL
 *
 *  Description détaillée de la constante symbolique NBRE_MAXIMAL
 */
#define  NBRE_MAXIMAL   (40)
```

4.3.5 - LES TYPES UTILISATEURS

```
/** @typedef byte
 *  @brief définition du type utilisateur byte
 *
 *  Description détaillée du type utilisateur byte
 */
typedef unsigned char   byte ;
```

4.3.6 - LES STRUCTURES

```
/** @struct _TMaStructure
 *  @brief Description rapide de la structure _TMaStructure
 *
 *  Description détaillée de la structure _TMaStructure
 */
struct _TMaStructure {
                    /** @brief Description rapide de nToto
                     *
                     *  Description détaillée de nToto
                     */
                    int nToto ;
                    int nTiti ; ///< Description rapide de nTiti
                    /** Description détaillée de nTutu
                     */
                    int nTutu ;
                } ;
```

Éventuellement à compléter à la suite par la définition d'un type utilisateur :

```
/** @typedef TMaStructure
 *  @brief Redéfinition du type [_TMaStructure](@ref _TMaStructure)
 */
typedef struct _TMaStructure   TMaStructure ;
```

4.3.7 - LES ÉNUMÉRATIONS

```
/** @enum   _TNombre
 *  @brief  Description rapide du type énuméré _TNombre
 *
 *  Description détaillée du type énuméré _TNombre
 */
enum _TNombre {
                /** @brief Description rapide de l'élément UN de l'énumération
                 *
                 *  Description détaillée de l'élément UN de l'énumération
                 */
                UN    = 1 ,
                DEUX  = 2 ,    ///< Description rapide de l'élément DEUX de l'énumération
                /** Description détaillée de l'élément TROIS de l'énumération
                 */
                TROIS = 3
      } ;
```

Éventuellement à compléter à la suite par la définition d'un type utilisateur :

```
/** @typedef  TNombre
 *  @brief Redéfinition du type [_TNombre](@ref _TNombre)
 */
typedef enum _TNombre   TNombre ;
```

4.3.8 - LES UNIONS

```
/** @union _TMonUnion
 *  @brief Description rapide de l'union _TMonUnion
 *
 *  Description détaillée de l'union _TMonUnion
 */
```

```
union _TMonUnion {
                  /** Description détaillée de fLong */
                  long fLong ;
                  /** @brief Description rapide de nEntier
                   *
                   *  Description détaillée de nEntier
                   */
                  int nEntier[2] ;
                  byte byOctet[4] ;      ///< Description rapide de byOctet
         } ;
```

Éventuellement à compléter à la suite par la définition d'un type utilisateur :

```
/** @typedef   TMonUnion
 *  @brief Redéfinition du type [_TMonUnion](@ref _TMonUnion)
 */
typedef union _TMonUnion   TMonUnion ;
```

4.3.9 - LES CLASSES

Chaque classe sera documentée dans son fichier en-tête, juste avant sa déclaration.

Le commentaire de documentation est assez succinct. en effet, dOxygen va se charger de générer les documentations des membres de la classe, de retrouver les divers types et classes utilisés et de faire le lien avec tout ça.

```
/** @brief Description rapide de MaClasse
 *
 *  Description détaillée de MaClasse.
 *
 *  @test   Voir la [procédure de test](chemin/NomFicheDeTest).
 *  @see    AutreClasse
 */
class MaClasse
{
public :
   // CONSTRUCTEURs et DESTRUCTEURs
   /// Description rapide du constructeur
   MaClasse() ;
   /// Description rapide du destructeur
   ~MaClasse() ;

   // METHODEs par ordre alphabétique ou groupe de fonctions logique -----------
   // ATTRIBUTs par ordre alphabétique ou groupe de fonctions logique ----------

protected :
   // METHODEs par ordre alphabétique ou groupe de fonctions logique -----------
   // ATTRIBUTs par ordre alphabétique ou groupe de fonctions logique ----------

private :
   // METHODEs par ordre alphabétique ou groupe de fonctions logique -----------
   // ATTRIBUTs par ordre alphabétique ou groupe de fonctions logique ----------

} ;
```

4.3.9.1 - LES ATTRIBUTS, CHAMPS DE STRUCTURE ET D'ÉNUMÉRATIONS

Sur les lignes précédentes la déclaration du membre, dans le fichier en-tête, ajoutez, selon le cas :

```
/** @brief Description rapide du membre
 *
 *  Description détaillée du membre
 */
```

ou

```
/** Description détaillée du membre
*/
```

ou

```
/** @brief Description rapide du membre */
```

ou

```
/// Description rapide du membre
```

Si vous préférez des commentaires en fin de ligne (à réserver aux descriptions rapides) :

```
/**< Description rapide du membre*/
```

ou encore

```
///< Description rapide du membre
```

4.3.9.2 - LES MÉTHODES

Il sera judicieux de placer la description rapide des méthodes dans le fichier .h correspondant :

```
/// Description rapide de la méthode AfficherAttributs
void AfficherAttributs(char * nomObjet) ;
```

La description détaillée sera de préférence dans le fichier .cpp :

```
/** Description détaillée de la méthode AfficherAttributs
 *
 * @pre               Description des préconditions nécessaires à la méthode
 *                    AfficherAttributs
 * @post              Description des postconditions nécessaires à la méthode
 *                    AfficherAttributs
 *
 * @param             Liste des paramètres de la méthode et de leur rôle
 * @param[in]         Liste des paramètres qui ne sont qu'en lecture dans la méthode,
 *                    leur rôle et plage d'entrée
 * @param[out]        Liste des paramètres qui seront modifiés par de la méthode,
 *                    leur rôle et valeurs retournées
 * @param[in,out]     Liste des paramètres qui seront modifiés par de la méthode,
 *                    leur rôle, plage d'entrée et valeurs retournées
 *
 * @return            Explication de la valeur de retour de la fonction
 *
 * @note              Remarques simples sur la méthode AfficherAttributs
 * @warning           Remarques importantes sur la méthode AfficherAttributs
 * @attention         Règles à respecter pour utiliser la méthode AfficherAttributs
 *
 * @test              Voir la [fiche de test](chemin relatif/NomFicheDeTest).
 *
 * @see               Liste des autres méthodes, variables et attributs en relation
 *                    avec la méthodeAfficherAttributs
*/
```

Si certains items n'ont pas lieu d'être (`@return`, `@param`...) il vous faudra les supprimer.

Par exemple :

```
/** Affichage des valeurs des attributs de l'objet
 *  @pre        L'objet est créé et ses attributs correctement initialisés
 *  @post       Les valeurs des attributs sont affichées sur la console
 *  @param[in]  sNomObjet   Le nom de variable, au format texte, de l'objet concerné
 *  @test       Voir la [fiche de test](../Fiches/AfficherAttributs.pdf).
 *  @see        InitialiserAttributs
 */
void MaClasse::afficherAttributs(char * sNomObjet)
{
    // Contenu de la méthode
}
```

5 - EXTRAS

5.1 - AMÉLIORATION DES DIAGRAMMES

dOxyGen possède un générateur de diagramme de classes intégré, mais n'est pas capable de créer des diagrammes de collaboration.

Vous pouvez supplanter cette fonctionnalité par un autre générateur graphique GraphVis.

Récupérez la dernière version sur le site de l'éditeur http://graphviz.org/Download_windows.php. Dans mon cas, il s'agit de la version 7.0.5.

Vous pouvez télécharger soit la version installable, qui s'installe facilement, soit la version zippée, qui nécessite une manipulation du système pour être opérationnelle :

Décompressez l'archive, par exemple dans le directory `C:\Program Files\graphviz`.

Accédez aux paramètres avancés du système par un clic droit sur le poste de travail de Windows, choisissez `Propriétés` dans le menu contextuel, puis l'option `Paramètres système avancés` dans le bandeau de gauche.

Cliquez sur le bouton `Variables d'environnement` pour accéder à la configuration du PATH.

Recherchez la ligne `Path`, dans la partie basse `Variables système`. Modifiez le `Path` en y ajoutant à la fin :

```
; C:\Program Files (x86)\graphviz\bin
```

Attention, le point virgule est important !

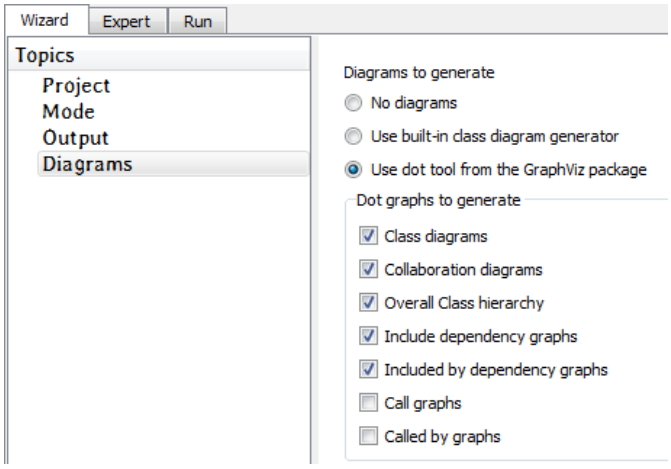
Validez.

GraphViz est opérationnel.

Vous pouvez maintenant avoir de plus beaux diagrammes de classes dans votre documentation, mais surtout les diagrammes de collaboration seront générés.

Par le dOxyWizard, modifiez le `Doxyfile.cfg` pour utiliser cette fonctionnalité.

Sélectionnez, en mode `Wizard`, l'option `Use dot tools from the GraphViz package` du topic `Diagrams`, plutôt que l'option par défaut `Use built-in class diagram generator` :



Vous pouvez ré-générer votre documentation dès que la modification sera sauvegardée.