Annexe 3 : Générateur de documentation Doxygen

Sommaire :

I - Introduction

Doxygen est un logiciel informatique libre permettant de créer de la documentation à partir du code source d'un programme. Pour cela, il tient compte de la grammaire du langage dans lequel est écrit le code source, ainsi que des commentaires s'ils sont écrits dans un format particulier.

Doxygen est capable d'analyser des fichiers sources écrits dans les langages **C**, **C++**, **Java**, Objective C, Python, IDL, VHDL et dans une certaine mesure PHP, C# et D.
La documentation peut être générée dans l'un ou plusieurs des formats suivants : **HTML** (compressé ou non), LaTeX, **RTF**, PostScript, **PDF** avec hyperliens, et **XML**.

II - Tags Javadoc

Pour commenter une application en vue de la génération de sa documentation avec **Doxygen**, il faut utiliser des **tags JavaDoc** présentés ci-dessous. Un **tag** commence toujours par @ ou \.

Tag Doxygen	**Description**
@a	Utilisé pour faire ressortir un paramètre dans sa description.
@attention	Message important à mettre en relief.
@author	Nom de l'auteur d'une méthode, classe, etc.
@b	Utilisé pour mettre en gras un mot dans une description; équivalent à **mot** en HTML.
@brief	Description brève du comportement d'une méthode, classe, etc.
@bug	Indique la présence d'un bug.
@class <nom> [<fichier-header>] [<chemin-du-header>]	Utilisé pour décrire une classe. Le premier paramètre est obligatoire, les suivants optionnels. Attention toutefois à ce que les paramètres soit corrects, la casse est prise en compte.
@date	Indique la date de création.

@def	Utilisé pour indiquer un commentaire pour une macro **#define**.
@details	Description détaillée du comportement d'une méthode, classe, etc.
@enum <nom>	Utilisé pour introduire la description d'une énumération de type **enum**.
@exception	Description du traitement d'une exception.
@file <nom>	Utilisé pour décrire un fichier. L'attribut <nom> doit être exact et fourni avec l'extension.
@fn	Utilisé pour introduire la description d'une fonction. Peut-être omis si la description est placée immédiatement sous la déclaration de la fonction et sans saut de ligne.
@include	Insère le code d'un fichier externe.
@package	Indique un nom de paquet.
@param	Indique le type, le nom et la description d'un paramètre d'une méthode.
@return	Indique le type et la description de la **valeur de retour** d'une méthode.
@see	Indique à l'utilisateur les autres méthodes, classes, etc à « **Voir également** ».
@struct <nom> [<fichier-header>] [<chemin-du-header>]	Utilisé pour décrire une **structure**. Le premier paramètre est obligatoire, les suivants optionnels. Attention toutefois à ce que les paramètres soient corrects, la casse est prise en compte.
@todo	Liste d'actions restant à réaliser.
@typedef	Utilisé pour introduire la description d'un **typedef**.
@union <nom> [<fichier-header>] [<chemin-du-header>]	Utilisé pour décrire une **union**. Le premier paramètre est obligatoire, les suivants optionnels. Attention toutefois à ce que les paramètres soient corrects, la casse est prise en compte.
@version	Version de l'application.

Ces doivent **tags** être utilisés dans des blocs de commentaires débutant par **/**** et finissant par ***/**. Les blocs de commentaires qui comportent plusieurs lignes devront comporter le caractère * en début de ligne.

III - Génération de la documentation avec Doxygen

III.1. Exemple

Voici un exemple comportant 3 fichiers :

- **controle.h** : la déclaration de la classe **Controle** (classe permettant de contrôler une trame d'octets en calculant un champ de contrôle);
- **controle.cpp** : la définition de la classe **Controle**;
- **main.cpp** : le test de la classe contrôle.

// **main.cpp**

```
/**
* @file    main.cpp
* @author  JCC
* @version 1.0
* @date    28 Fevrier 2014
* @brief   Test de la Classe Controle.
*          Calcul du champ de controle d'une trame de 5 octets.
* @see Controle
*/
#include "controle.h"
int main(void)
    {
    Controle *monControle;     // objet monControle
    monControle = new Controle();
    ULONG crc;     // variable recevant le controle
    BYTE buffer[] = {0x00,0x01,0x02,0x03,0x04};    // Trame à traiter
    int taille = 5;
    crc = monControle->GetCrc( buffer, taille );     // calcul du crc
    printf( "crc =  0x%x\n", crc );     // affichage du crc
    }
```

// **controle.h**

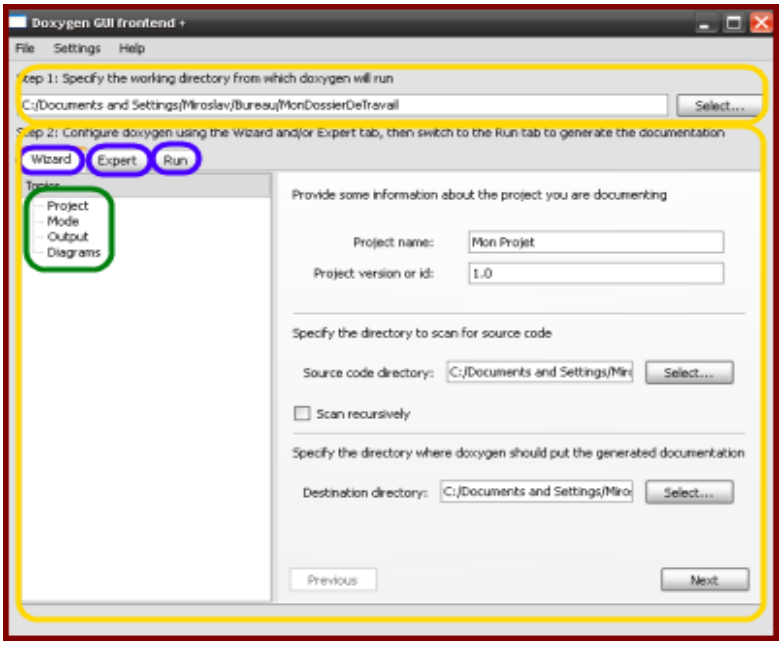
```
/**
* @file    controle.h
* @author  JCC
* @version 1.0
* @date    28 Fevrier 2014
* @brief   Déclaration de la classe Controle.
*          Cette classe permet de controler une trame d'octets.
*/

#ifndef _CONTROLE_H
#define _CONTROLE_H
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/**
* @def Type ULONG
*/
#define ULONG       unsigned int          /*!<   entier   non   signé
codé sur 16 bits (mot) */
/**
* @def Type BYTE
*/
#define BYTE        unsigned char         /*!<   entier   non   signé
codé sur 8 bits (octet) */
/**
* @brief Classe Controle
*/
 class Controle
 {
 private :
 public:
            Controle();
            ULONG GetCrc( BYTE *trame, ULONG taille );
 };
#endif
```

// **controle.cpp**

```
/**
* @file    controle.cpp
* @author  JCC
* @version 1.0
* @date    28 Fevrier 2014
* @brief   Définition de la classe Controle.
*          Cette classe permet de controler une trame d'octets.
*/
#include "controle.h"
/**
* @brief   Constructeur de la classe Controle.
* @param   Néant
* @return  \e Néant
* @see Controle#GetCrc
*/
 Controle::Controle()
 {
 }
/**
* @brief   Méthode GetCrc  de la classe Controle.
*          Calcule le champ de controle d'une trame d'octets.
* @param   trame
*          La trame à analyser
* @param   taille
*          La taille de la trame à analyser
* @return  Un \e ULONG représentant Le champ de controle codé
sur 16 bits.
* @see Controle#Controle
*/
 ULONG Controle::GetCrc( BYTE *trame, ULONG taille)
 {
 ULONG crc = 0x0000;
 int i=0;
 while (i < taille)
            {
            crc = crc + trame[i];
            i++;
            }
 return (crc);
 }
```

III.2. Génération de la documentation

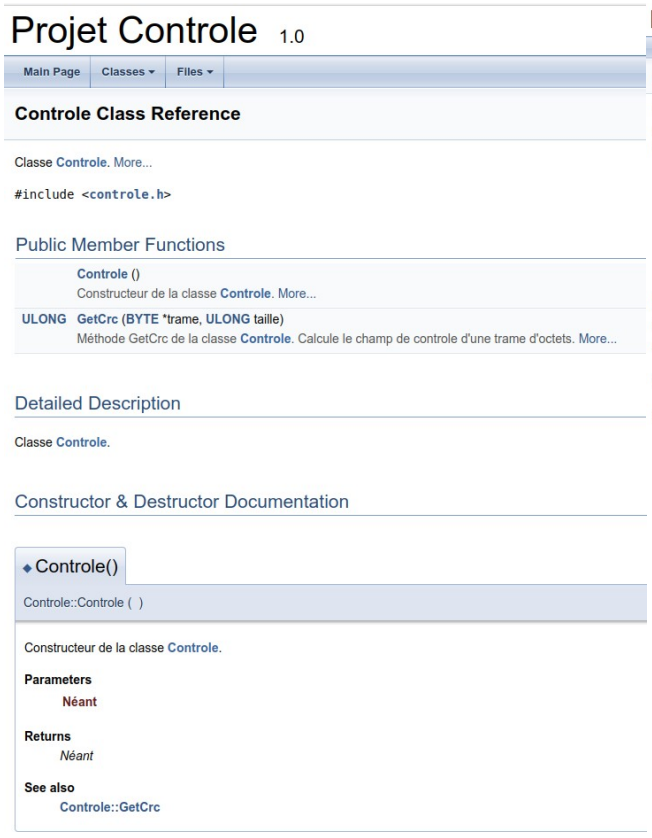
Pour utiliser **Doxygen** sur un poste **Linux** il faut installer les paquets **doxygen**, **doxygen-gui** et **doxygen-doc** : **apt-get install doxygen doxygen-gui doxygen-doc**. On peut maintenant utiliser **Doxygen** en lançant la commande **doxywizard** depuis un terminal.

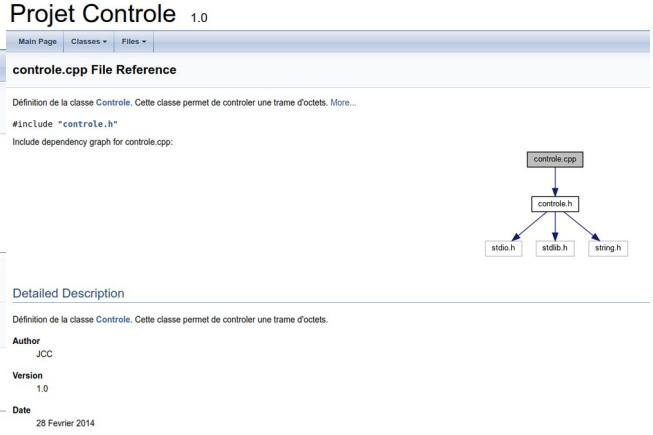


Il suffit de préciser le chemin vers le répertoire de travail, remplir les 4 parties de l'assistant **Wizard** de **Doxygen**. Outre le mode **Wizard**, on remarque que le logiciel propose un mode **Expert**. L'onglet **Run** sert à lancer la génération de la documentation une fois que la configuration est correctement effectuée, soit par l'onglet **Wizard** soit par l'onglet **Expert**.

III.3. Résultat au format html

Cela donne le résultat suivant au format **HTML** :





III.4. Résultat au format pdf

Pour générer la documentation au format **pdf**, il suffit d'aller dans le dossier **latex** et de taper la commande : **make pdf**.

Si un message d'erreur apparaît, il faudra installer les paquets suivants : **sudo apt-get install texlive-latex-base texmaker texlive-fonts-recommended texlive-science texlive-publishers texlive-fonts-extra texlive-science-doc texlive-science**.

Il suffit maintenant de visualiser le contenu du fichier **refman.pdf** dont voici un extrait (le sommaire) :

Contents