Table des matières

Documentation du code avec Doxygen

Doxygen

Doxygen est un système de documentation pour C, C++, Java, Python, Php et autres langages. Il permet de générer la documentation de vos développements :

- à partir des commentaires insérés dans le code source
- à défaut de commentaires, à partir de la structure du code lui même. La documentation générée sera dans ce cas minimale.

La documentation peut être produite dans des formats variés tels que du HTML, du Latex, du RTF ou du XML.

Doxygen est un logiciel libre, publié sous licence GPL V2.0.

Liens :

- Site officiel
- Documentation
- FAQ

Installation

Ubuntu :

```
$ sudo apt-get install doxygen doxygen-gui doxygen-doc
```

Pour les graphiques :

```
$ sudo apt-get install graphviz
```

Voir aussi : doc.ubuntu-fr.org/doxygen

Utilisation

Pour lancer l'interface graphique de Doxygen, ouvrez un terminal et entrez la commande suivante :

```
$ doxywizard &
```

L'onglet **Wizard** vous permettra :

- de créer votre projet
- de sélectionner le dossier contenant les sources ou celui accueillant votre documentation
- de sélectionner le format de sortie : HTML avec ou sans frames, Latex, RTF, pages man, XML, PDF, Postscript.
- de générer des diagrammes

L'onglet **Expert** vous permettra d'accéder aux options avancées.

Il ne vous reste alors plus qu'à cliquer sur **Run** pour obtenir le résultat. Mais encore faut-il avoir documenter son code ?

Principe : Doxygen nous amène à distinguer deux types de commentaires.

— Les commentaires "privés" : ces commentaires sont destinés aux développeurs et restent dans le code source. Ils ne seront donc pas extraits par Doxygen.

```
// Un commentaire privé sur une seule ligne

/*
 * Un commentaire privé sur plusieurs lignes
 */
```

— Les commentaires "publics" : ces commentaires sont destinés à la documentation et seront donc extraits par Doxygen.

```
#define NB 42 //!< Un nombre NB

/**
 * Une fonction foo
 */
void foo();
```

Remarque : Doxygen propose plusieurs syntaxes.

Doxygen est capable d'extraire tous les identifiants de votre code (variables, attributs, fonctions, méthodes, structures, classes, ...) et il ne reste plus qu'à les commenter. Pour préciser le type d'informations à fournir pour la documentation, Doxygen utilise un ensemble de *tags* (ou commandes) préfixés par `@` ou `\` :

```
/**
 * @file        exemple.h
 * @brief       Contient la déclaration de la classe Exemple
 * @details     La classe \c Exemple permet de montrer l'utilisation des \em tags \b Doxygen
 * @author      Thierry vaira <thierr.vaira@gmail.com>
 * @version     0.1
 * @date        2020
 * @copyright   GNU Public License.
 */
```

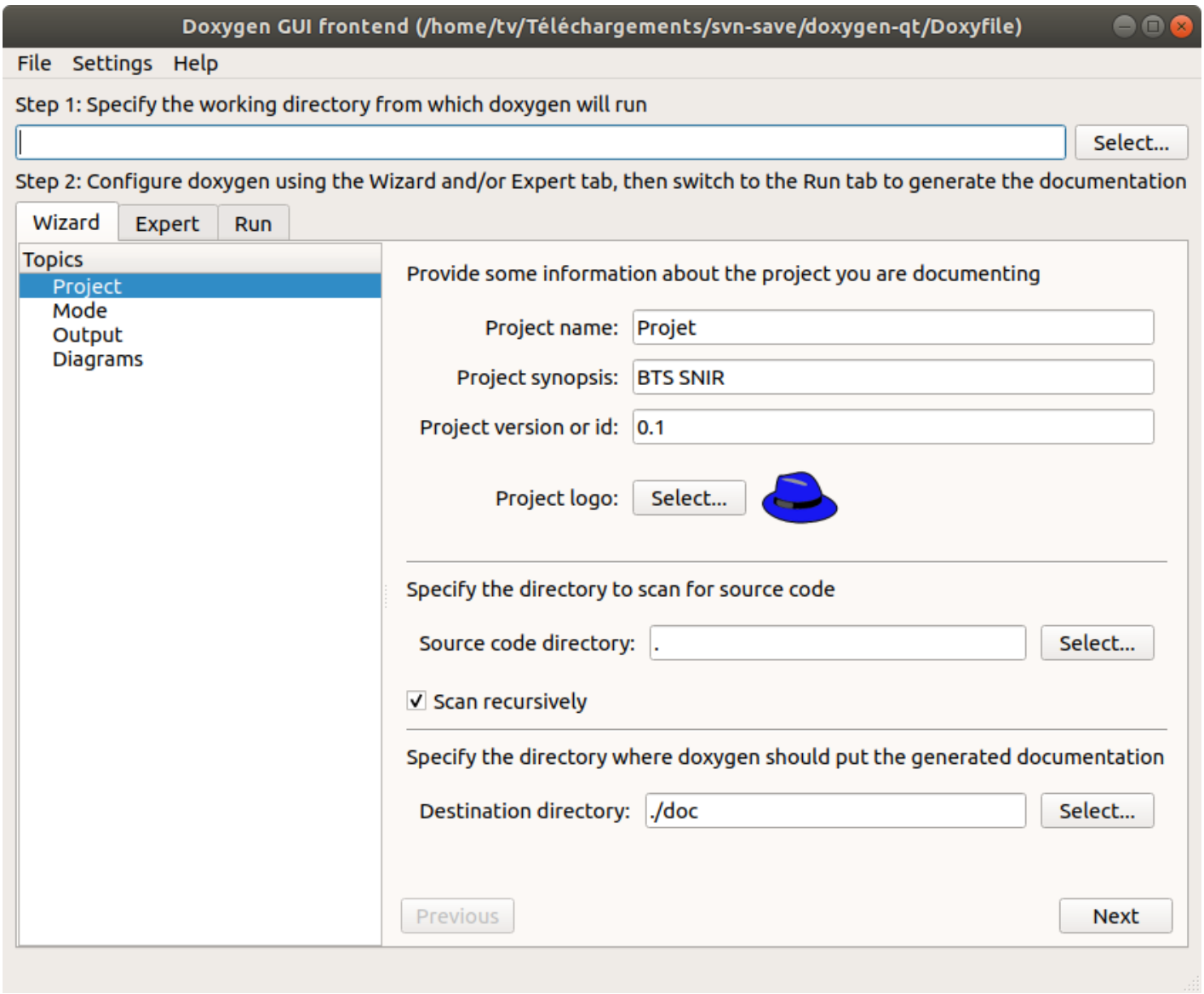
Voir aussi : Liste des commandes

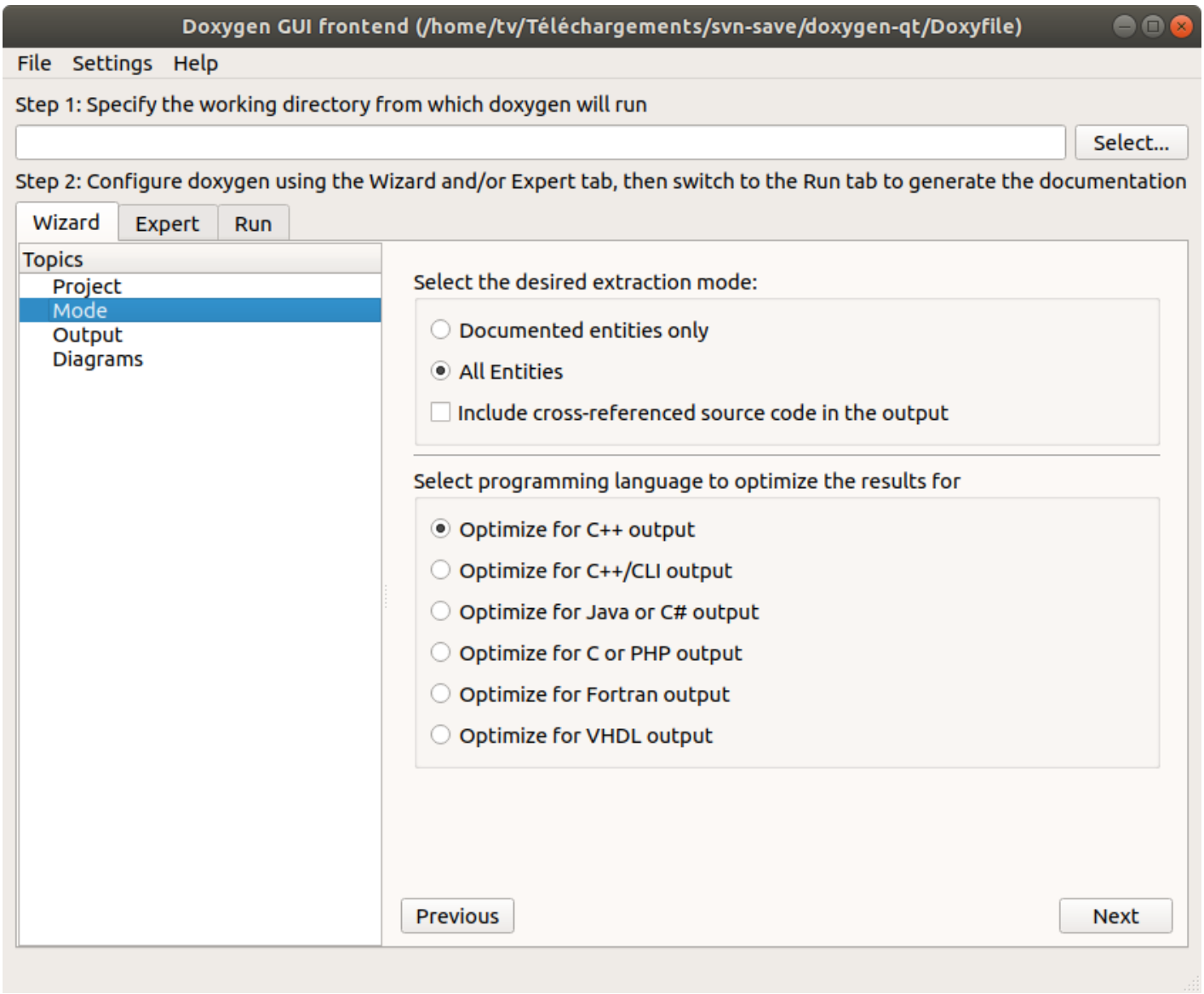
Exemple de configuration

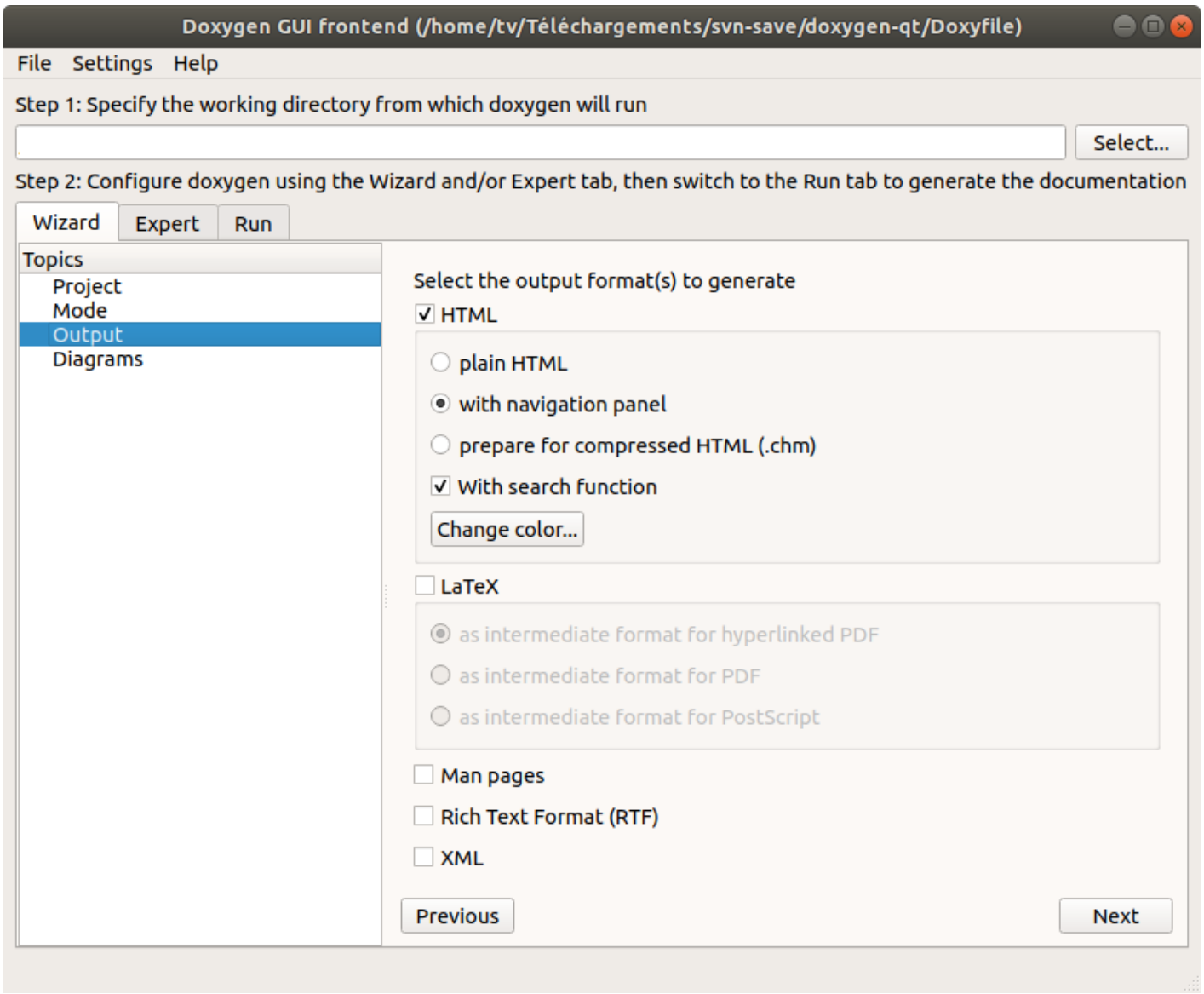
Remarques :

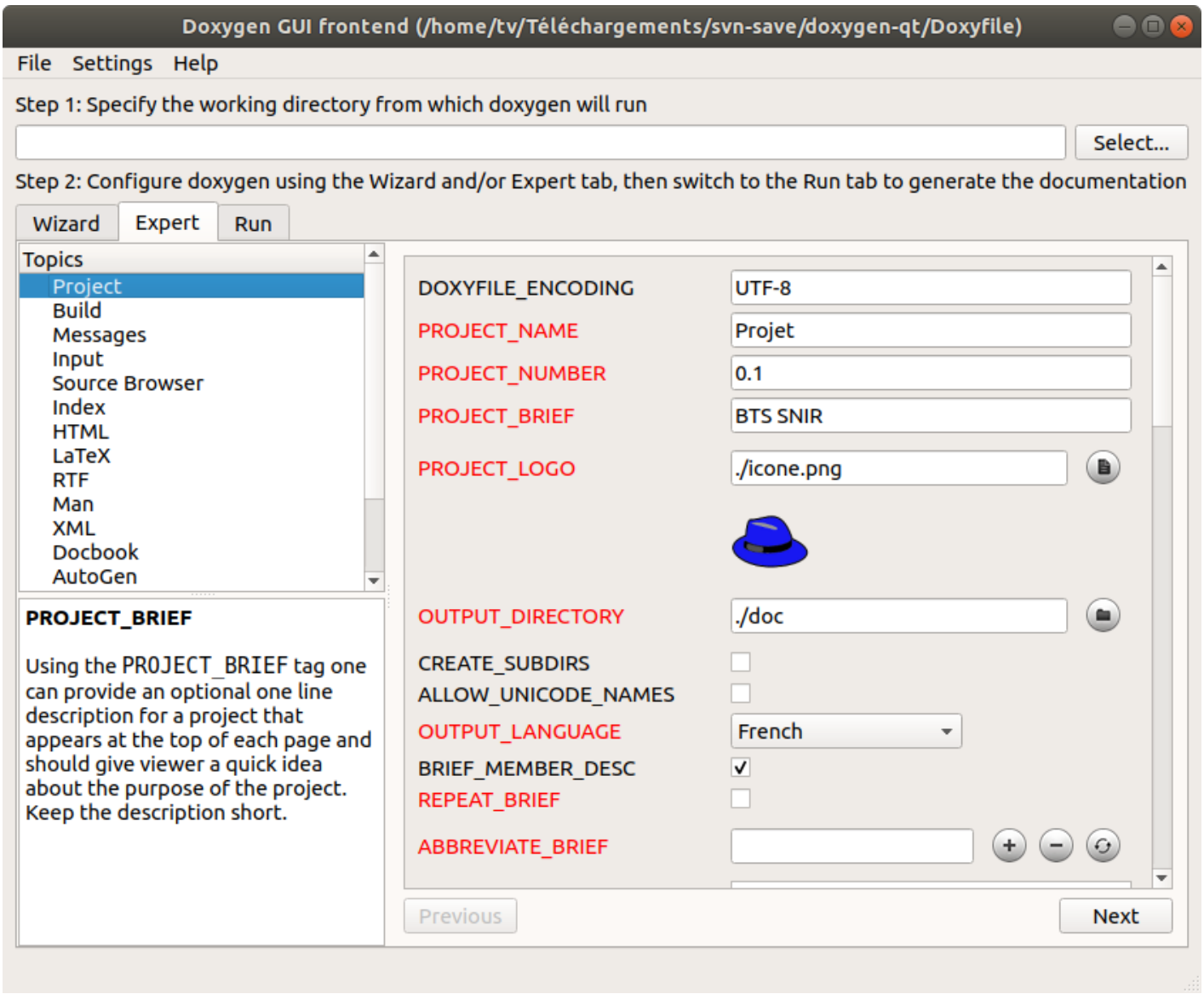
— Sélectionner le Français pour `OUTPUT_LANGUAGE`
— Le format HTML est adapté à la navigation et le format PDF est indispensable pour imprimer et avoir l'ensemble de la documentation dans un seul fichier. Le format RTF sera utile pour y faire des copier/coller pour votre dossier.

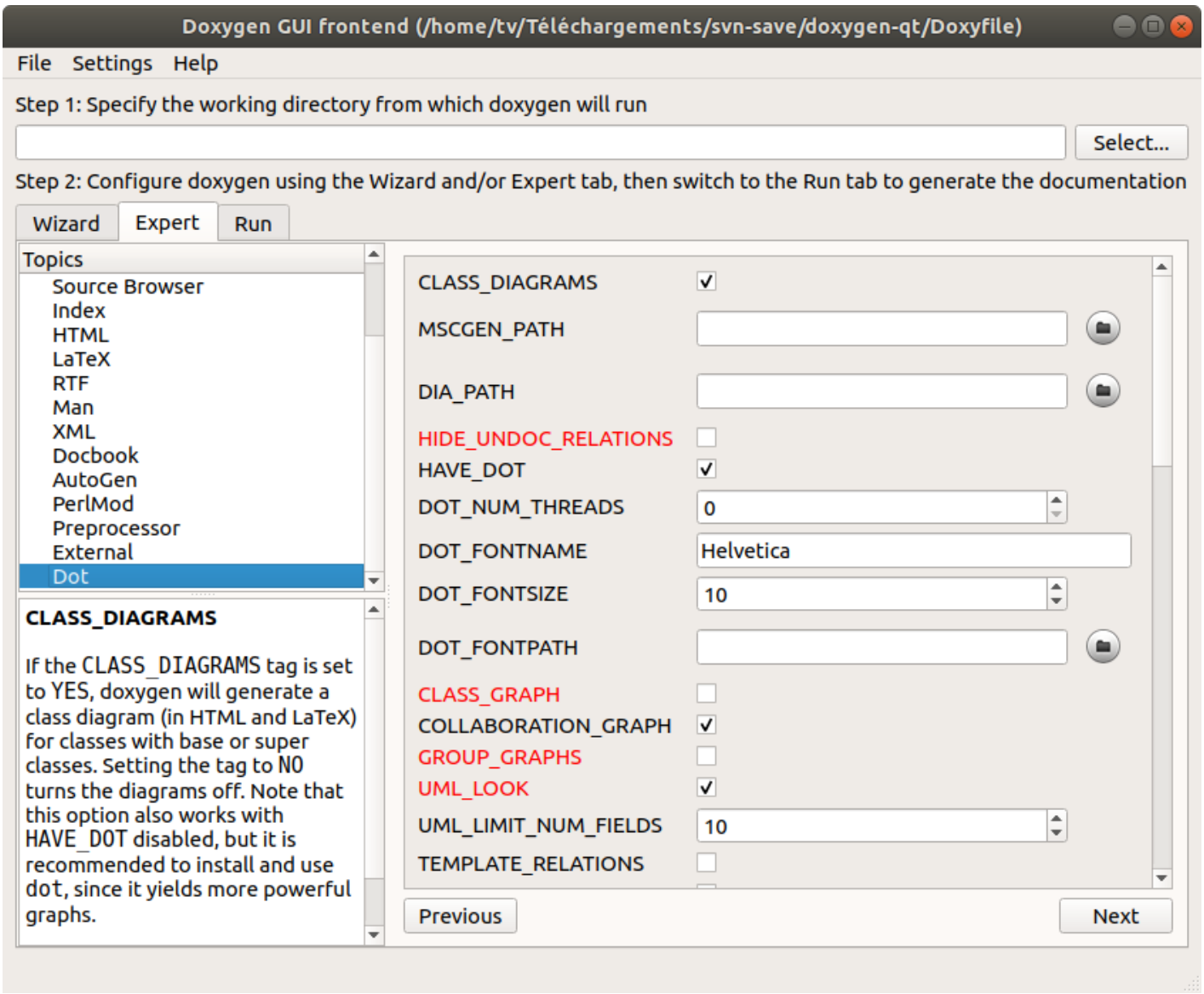
Quelques options de configuration :

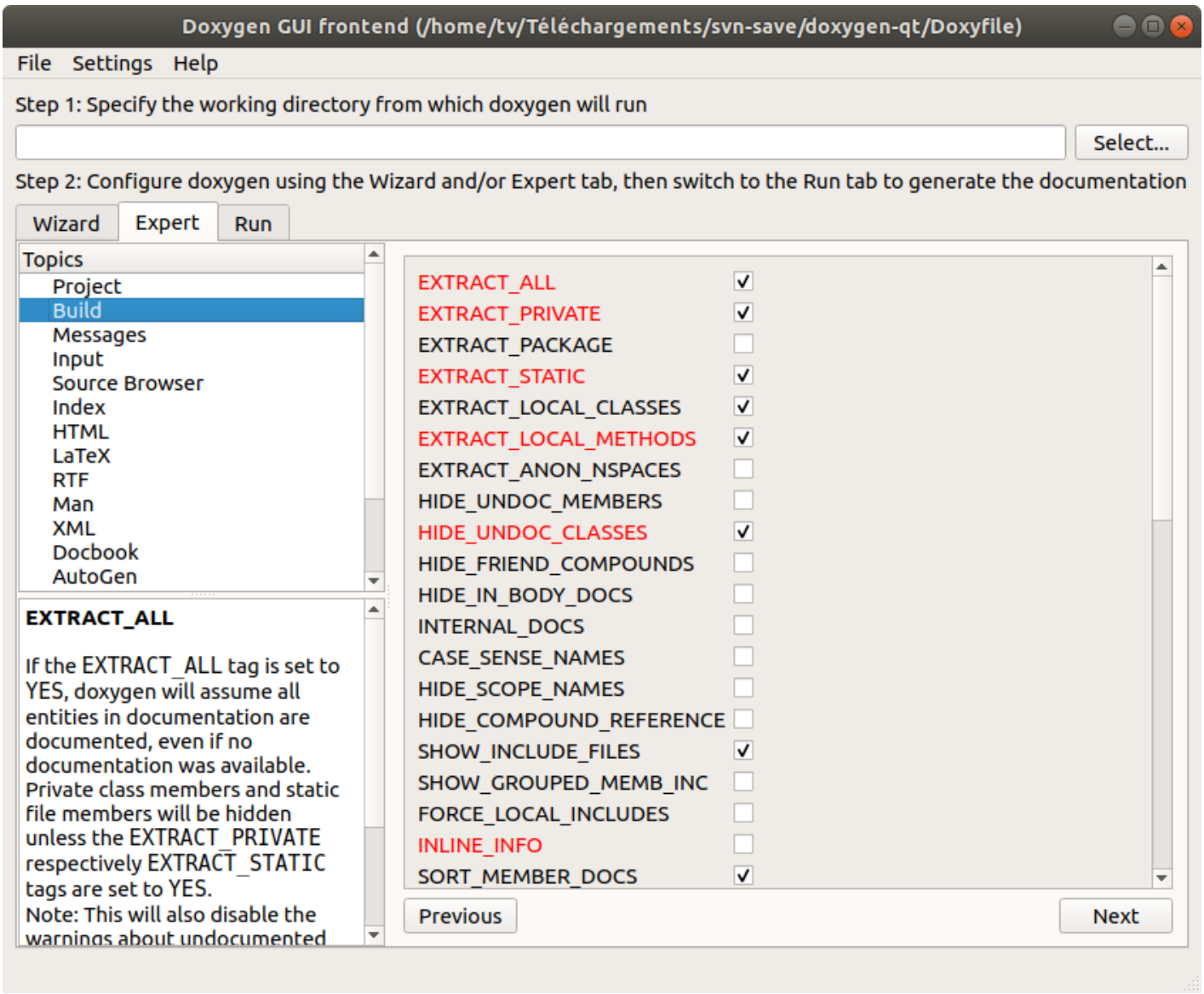

Doxygen GUI frontend (/home/tv/Téléchargements/svn-save/doxygen-qt/Doxyfile)
File
Settings
Help
Step 1: Specify the working directory from which doxygen will run
Select...
Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation
Wizard
Expert
Run
Topics
Project
Mode
Output
Diagrams
Provide some information about the project you are documenting
Project name:
Projet
Project synopsis:
BTS SNIR
Project version or id:
0.1
Project logo:
Select...
Specify the directory to scan for source code
Source code directory:
.
Select...
Scan recursively
Specify the directory where doxygen should put the generated documentation
Destination directory:
./doc
Select...
Previous
Next

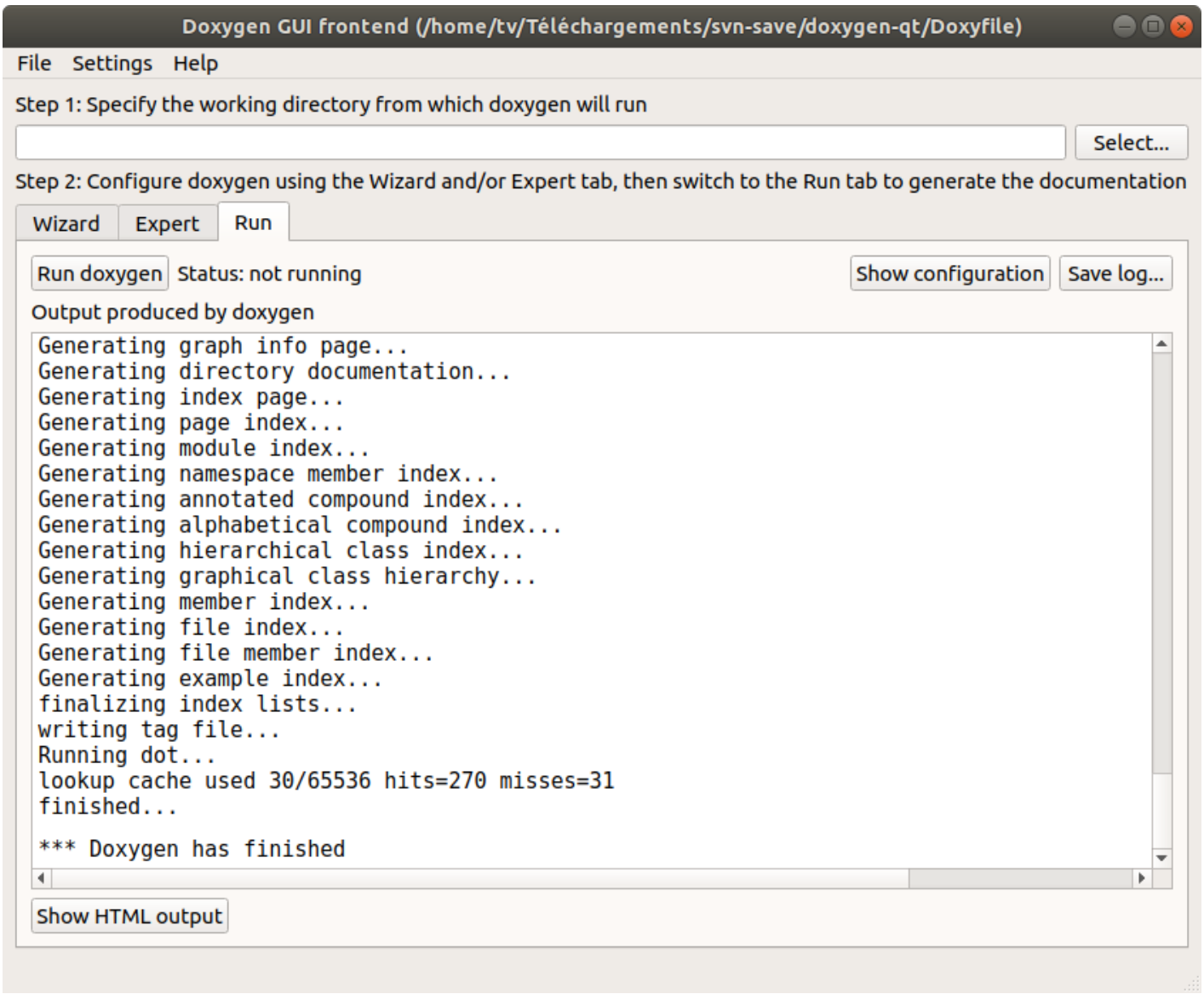

Doxygen GUI frontend (/home/tv/Téléchargements/svn-save/doxygen-qt/Doxyfile)
File
Settings
Help
Step 1: Specify the working directory from which doxygen will run
Select...
Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation
Wizard
Expert
Run
Topics
Project
Mode
Output
Diagrams
Select the desired extraction mode:
Documented entities only
All Entities
Include cross-referenced source code in the output
Select programming language to optimize the results for
Optimize for C++ output
Optimize for C++/CLI output
Optimize for Java or C# output
Optimize for C or PHP output
Optimize for Fortran output
Optimize for VHDL output
Previous
Next


Doxygen GUI frontend (/home/tv/Téléchargements/svn-save/doxygen-qt/Doxyfile)
File
Settings
Help
Step 1: Specify the working directory from which doxygen will run
Select...
Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation
Wizard
Expert
Run
Topics
Project
Mode
Output
Diagrams
Select the output format(s) to generate
HTML
plain HTML
with navigation panel
prepare for compressed HTML (.chm)
With search function
Change color...
LaTeX
as intermediate format for hyperlinked PDF
as intermediate format for PDF
as intermediate format for PostScript
Man pages
Rich Text Format (RTF)
XML
Previous
Next

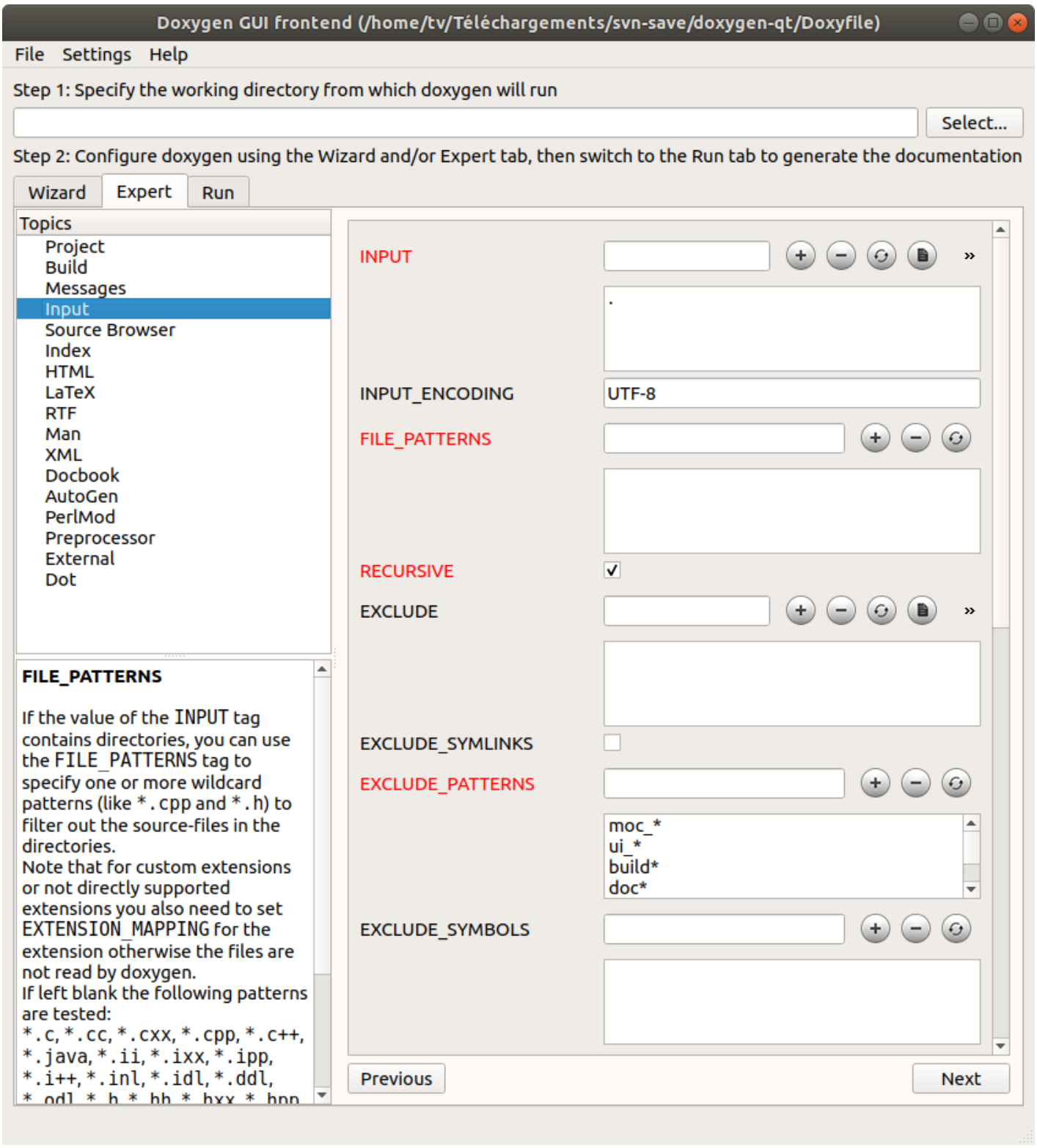

Doxygen GUI frontend (/home/tv/Téléchargements/svn-save/doxygen-qt/Doxyfile)
File Settings Help
Step 1: Specify the working directory from which doxygen will run
Select...
Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation
Wizard Expert Run
Topics
Project
Build
Messages
Input
Source Browser
Index
HTML
LaTeX
RTF
Man
XML
Docbook
AutoGen
PROJECT_BRIEF
Using the PROJECT_BRIEF tag one can provide an optional one line description for a project that appears at the top of each page and should give viewer a quick idea about the purpose of the project. Keep the description short.
DOXYFILE_ENCODING UTF-8
PROJECT_NAME Projet
PROJECT_NUMBER 0.1
PROJECT_BRIEF BTS SNIR
PROJECT_LOGO ./icone.png
OUTPUT_DIRECTORY ./doc
CREATE_SUBDIRS
ALLOW_UNICODE_NAMES
OUTPUT_LANGUAGE French
BRIEF_MEMBER_DESC
REPEAT_BRIEF
ABBREVIATE_BRIEF
Previous
Next


Doxygen GUI frontend (/home/tv/Téléchargements/svn-save/doxygen-qt/Doxyfile)
File
Settings
Help
Step 1: Specify the working directory from which doxygen will run
Select...
Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation
Wizard
Expert
Run
Topics
Source Browser
Index
HTML
LaTeX
RTF
Man
XML
Docbook
AutoGen
PerlMod
Preprocessor
External
Dot
CLASS_DIAGRAMS
If the CLASS_DIAGRAMS tag is set to YES, doxygen will generate a class diagram (in HTML and LaTeX) for classes with base or super classes. Setting the tag to NO turns the diagrams off. Note that this option also works with HAVE_DOT disabled, but it is recommended to install and use dot, since it yields more powerful graphs.
CLASS_DIAGRAMS
MSCGEN_PATH
DIA_PATH
HIDE_UNDOC_RELATIONS
HAVE_DOT
DOT_NUM_THREADS
0
DOT_FONTNAME
Helvetica
DOT_FONTSIZE
10
DOT_FONTPATH
CLASS_GRAPH
COLLABORATION_GRAPH
GROUP_GRAPHS
UML_LOOK
UML_LIMIT_NUM_FIELDS
10
TEMPLATE_RELATIONS
Previous
Next


Doxygen GUI frontend (/home/tv/Téléchargements/svn-save/doxygen-qt/Doxyfile)
File
Settings
Help
Step 1: Specify the working directory from which doxygen will run
Select...
Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation
Wizard
Expert
Run
Topics
Project
Build
Messages
Input
Source Browser
Index
HTML
LaTeX
RTF
Man
XML
Docbook
AutoGen
EXTRACT_ALL
If the EXTRACT_ALL tag is set to YES, doxygen will assume all entities in documentation are documented, even if no documentation was available. Private class members and static file members will be hidden unless the EXTRACT_PRIVATE respectively EXTRACT_STATIC tags are set to YES.
Note: This will also disable the
EXTRACT_ALL
EXTRACT_PRIVATE
EXTRACT_PACKAGE
EXTRACT_STATIC
EXTRACT_LOCAL_CLASSES
EXTRACT_LOCAL_METHODS
EXTRACT_ANON_NSPACES
HIDE_UNDOC_MEMBERS
HIDE_UNDOC_CLASSES
HIDE_FRIEND_COMPOUNDS
HIDE_IN_BODY_DOCS
INTERNAL_DOCS
CASE_SENSE_NAMES
HIDE_SCOPE_NAMES
HIDE_COMPOUND_REFERENCE
SHOW_INCLUDE_FILES
SHOW_GROUPED_MEMB_INC
FORCE_LOCAL_INCLUDES
INLINE_INFO
SORT_MEMBER_DOCS
Previous
Next



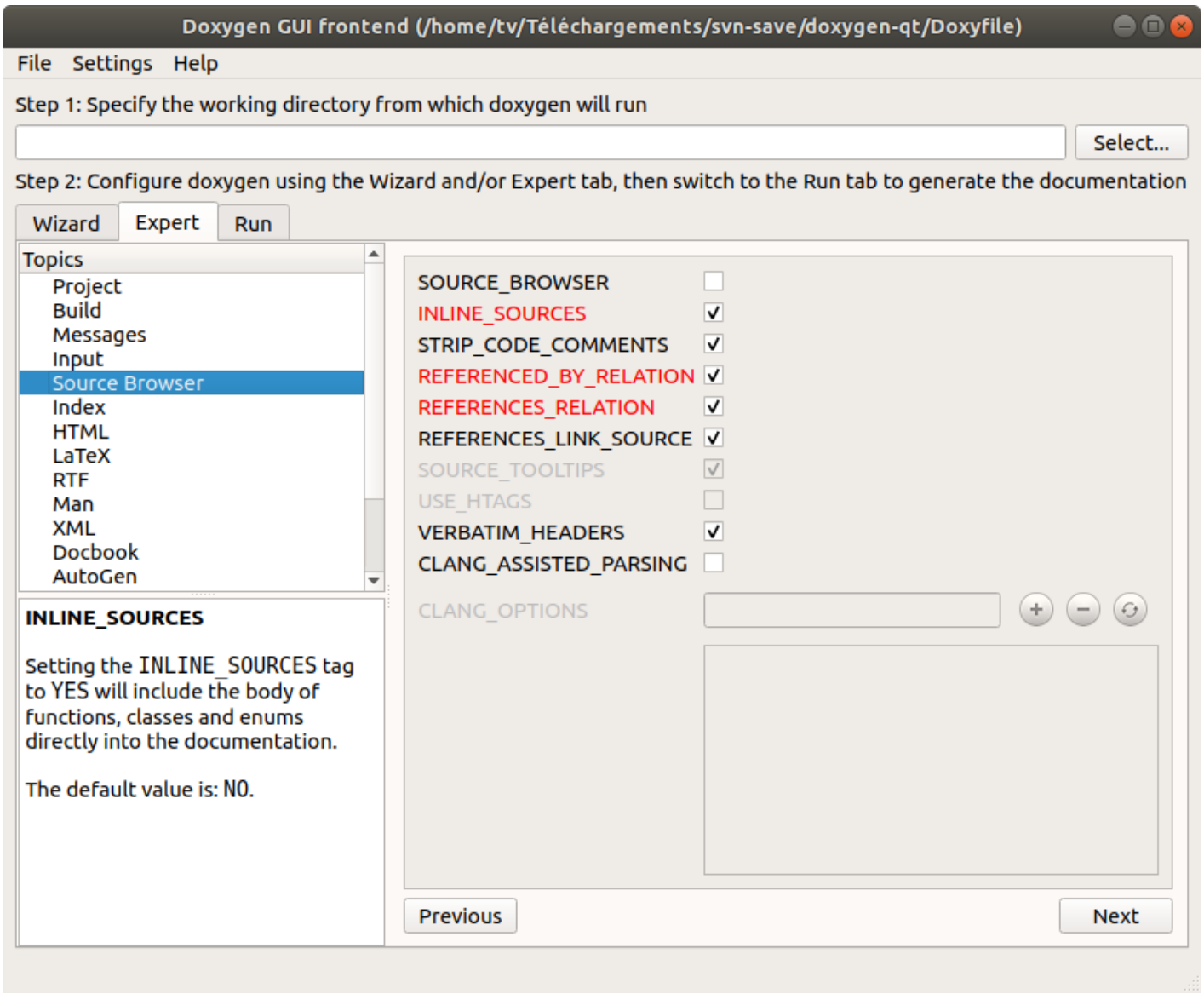
Ensuite, il faut indiquer les chemins vers les fichiers sources à inclure (et/ou à exclure) :



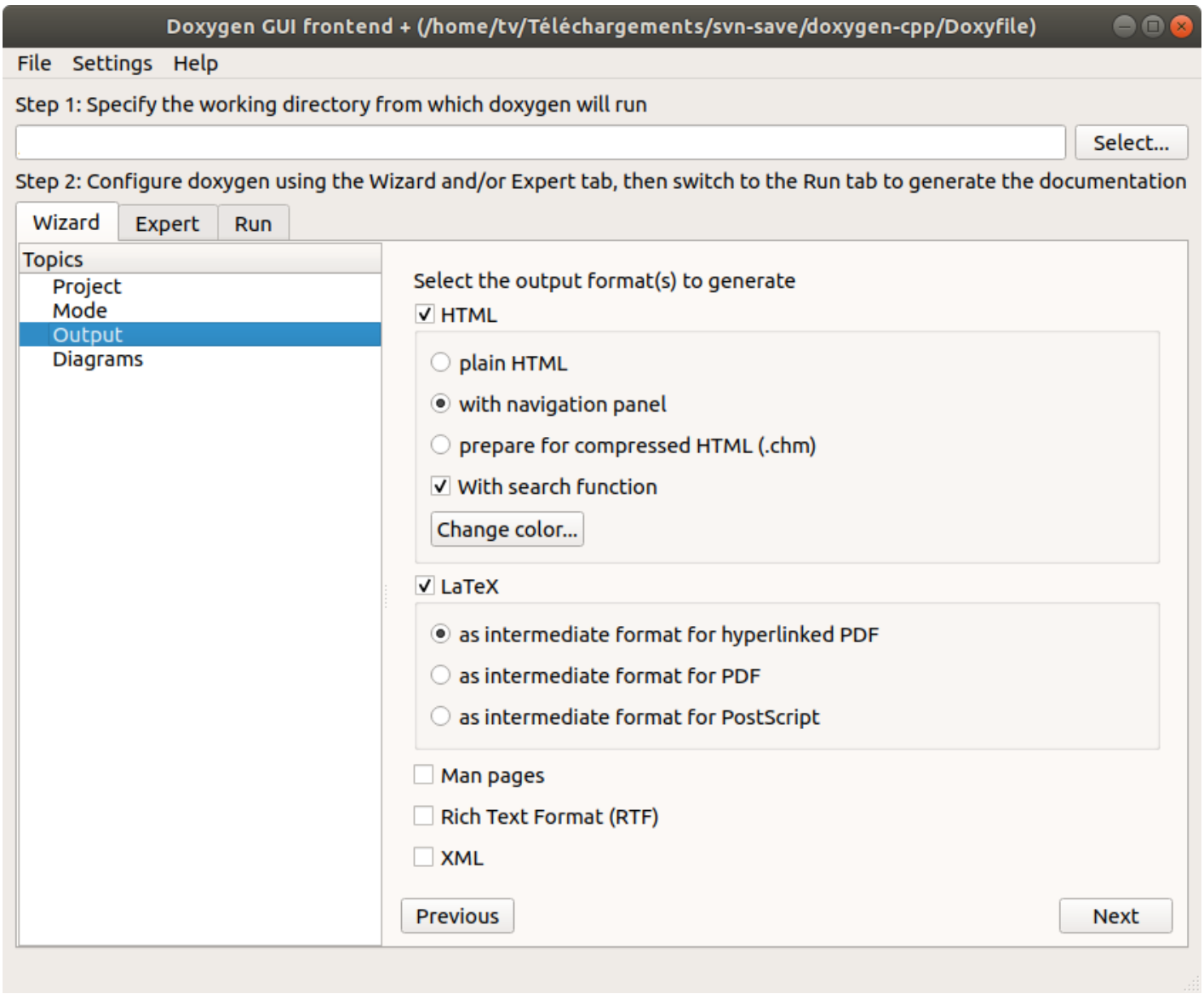
N'oubliez pas d'enregistrer votre configuration dans un fichier `Doxyfile`.

Remarque : il est intéressant de générer aussi les **diagrammes de classes**. Pour cela, veillez à sélectionner les options suivantes :

— Dans *Source Browser* :



Pour obtenir un document PDF `refman.pdf` :



Ensuite :

```
$ cd ./doc/latex/
$ make
```

Personnalisation du rendu

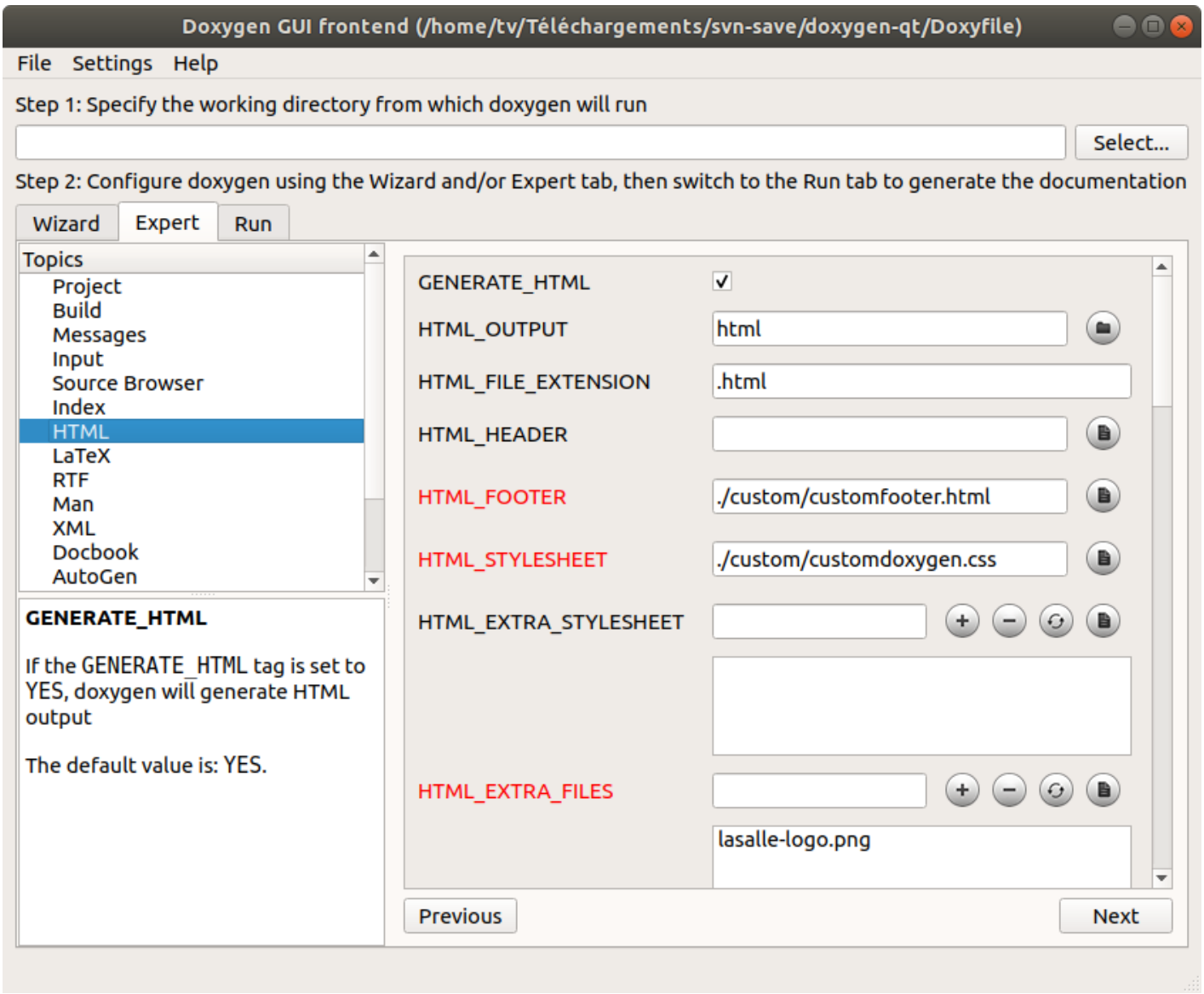
— **HTML** :

Il faut tout d'abord générer les fichiers par défaut d'entête (*header*) et de pied de page (*footer*) :

```
$ doxygen -w html header.html footer.html stylesheet.css
```

Ensuite vous pouvez les personnaliser et configurer le fichier `Doxyfile` :

```
HTML_HEADER            =
HTML_FOOTER            = ./custom/customfooter.html
HTML_STYLESHEET        = ./custom/customdoxygen.css
HTML_EXTRA_FILES       = lasalle-logo.png
```



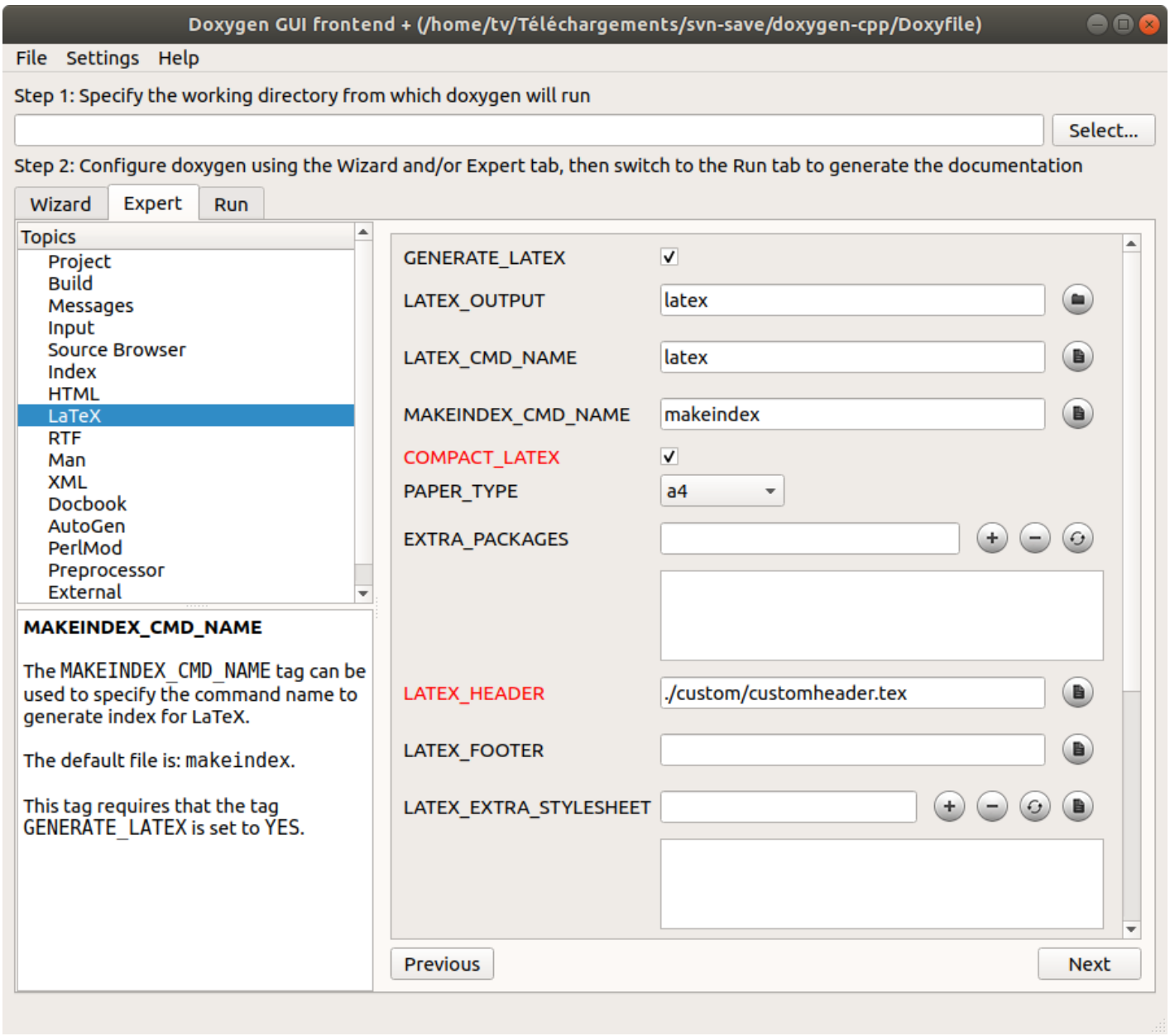
— **PDF** :

Il faut tout d'abord générer les fichiers par défaut d'entête (*header*) et de pied de page (*footer*) :

```
$ doxygen -w latex header.tex footer.tex doxygen.sty
```

Ensuite vous pouvez les personnaliser et configurer le fichier `Doxyfile` :

```
LATEX_HEADER           = ./custom/customheader.tex
LATEX_FOOTER           =
LATEX_EXTRA_STYLESHEET =
```



Il est aussi possible d'ajouter son *package* personnalisé :

```
EXTRA_PACKAGES        = custompackage
LATEX_EXTRA_FILES     = ./custom/custompackage.sty
```

Et compléter son *package* personnalisé :

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{custompackage}

...
```

Exemples

Exemples de fichier `Doxyfile` pour le projet : doxygen-projet-cpp.zip et doxygen-projet-java.zip

Exemple de rendu HTML : pour C++ ou pour Java

Exemple de rendu PDF : pour C++ ou pour Java

Exemple de rendu HTML : Revue 2 (projet BTS SNIR)

Remarque : Doxygen propose le tag `@fn` pour identifier une fonction ou une méthode. Il est inutile de le définir car Doxygen le fera automatiquement si le commentaire de documentation est placé devant la fonction ou méthode (comme indiqué dans les exemples ci-dessous).

— les constantes et/ou macros :

```
/**
 * @def NB
 * @brief Définit le nombre 42 !
*/
#define NB 42 //!< Un nombre NB
```

— les types énumérés :

```
/**
 * @enum TEnum
 * @brief Description du type énuméré ...
 *
 * @var TEnum Val1
 * @brief Description de Val1 ...
 */
enum TEnum //! Un type énuméré ...
{
    Val1,
    Val2    //!< Description de Val2 ...
};
```

— les structures :

```
/**
 * @struct Etat
 * @brief Structure ...
 *
 */
struct Etat
{
    bool present; //!< Membre définissant ...
};
```

— les classes :

```
/**
 * @class       Exemple exemple.h "exemple.h"
 * @brief       La déclaration de la classe Exemple
 * @details     La classe \c Exemple permet de montrer l'utilisation des \em tags \b Doxygen
 * @author      Thierry vaira <thierr.vaira@gmail.com>
 * @version     0.1
 * @date        2020
 * @note        Une note à l'attention de ceux qui lisent les notes
 * @pre         Initialisez d'abord le système
 * @post        L'objet est initialisé ou pas
 * @bug         La copie est impossible ou illégale
 * @warning     Une mauvaise utilisation peut faire planter votre application (c'est votre faute)
 * @attention   Il faut toujours faire attention
 * @remark      Une remarque à faire ?
 * @copyright   GNU Public License.
 */
class Exemple
{
};
```

— les attributs :

```
class Exemple
{
    private:
```

```
        int a; //!< a est ...
};
```

— les méthodes :

```
/**
 * @brief Constructeur par défaut de la classe Exemple.
 *
 * @see   Exemple::Exemple(int a)
 */
Exemple::Exemple() : a(0)
{
}

/**
 * @brief Constructeur de la classe Exemple.
 * @overload
 * @param a la valeur initiale de l'attribut a
 *
 * @see   Exemple::Exemple(int a)
 * @see   https://doc.qt.io/qt-5/qdatetime.html
 */
Exemple::Exemple(int a) : a(a)
{
    QDateTime maintenant = QDateTime::currentDateTime();
    qDebug() << Q_FUNC_INFO << maintenant.toString("dd/MM/yyyy") << maintenant.toString("hh:mm:ss") << "a" << a
    << this;
}

/**
 * @brief Accesseur de l'attribut a
 * @callergraph
 * @return a la valeur de l'attribut a
 * @retval int la valeur de l'attribut a
 */
int Exemple::getA() const
{
    return a;
}

/**
 * @brief Mutateur de l'attribut a
 * @callgraph
 * @param a ...
 * @exception range_error Si a est négatif
 */
void Exemple::setA(int a)
{
    if(a < 0 || a > NB)
        throw range_error("erreur plage");
    this->a = a;
    qDebug() << Q_FUNC_INFO << "a" << getA();
}

/**
 * @brief Montre le sens des paramètres
 * @param[in]       a1  ...
 * @param[out]      a2  ...
 * @param[in,out]   a3  ...
 */
void Exemple::copy(const int &a1, int &a2, int *a3)
{
    /**
     * @todo Implémenter la méthode
     */
}
```

— les fichiers *header* :

```
/**
 * @file exemple.h
 * @brief La déclaration de la classe Exemple
 * si besoin auteur, version et date
 */
```

— les fichiers d'implémentation :

```
/**
 * @file exemple.cpp
 * @brief La définition de la classe Exemple
 * si besoin auteur, version et date
 */
```

— le fichier principal (par exemple `main.cpp`) :

```
/**
 * @file main.cpp
 *
 * @brief Programme principal ...
 * @details Crée et affiche la fenêtre principale de l'application ...
 * @author ...
 * @author ...
 * @version ...
 *
 * @param argc
 * @param argv[]
 * @return int
 *
 */
```

— du code :

```
/*
 * ...
 * Instanciation :
 * \code{.cpp}
 * Exemple exemple1;
 * Exemple exemple2(5);
 * \endcode
 * \n
```

— des extraits de code utilisables avec `@snippet` :

```
int main()
{
    //! [Test]
    Exemple exemple1;
    Exemple exemple2(5);
    //! [Test]

    return 0;
}
```

Puis :

```
/**
 * @brief Constructeur par défaut de la classe Exemple.
 *
 * \b Tests :
 * @snippet ./test.cpp Test
 *
 */
Exemple::Exemple() : a(0)
{
}
```

— des exemples de fichier :

```
/**
 * ...
 * ...
 * @example test.cpp
 * @brief Test d'utilisation de la classe Exemple
 *
 */
class Exemple
{
};
```

— des graphiques :

```
/**
 *   \dot
 *  digraph example {
 *      node [shape=box, fontname=Helvetica, fontsize=12, color=black];
 *      a [ label="QObject" ];
 *      b [ label="Exemple" URL="\ref Exemple" fillcolor=lightblue,style=filled];
 *      a -> b [ arrowhead="normal", fillcolor=white,style=filled,dir=back ];
 *  }
 *  \enddot
 */
```

— des images :

```
/**
 * ...
 * \image html screenshot.png
 */
```

Remarque : il faut préciser le chemin des images avec la variable `IMAGE_PATH` dans le fichier `Doxyfile`.

Pages de documentation

Il est possible d'ajouter des **pages de documentation** : soit de simples fichiers soit des fichiers au format Markdown (lire aussi markdown-vscode.pdf).

— une page principale :

```
/*! \mainpage Page principale du projet XXX
 *
 * \tableofcontents
 *
 * \section section_intro Introduction
 *
 *
 * Bla bla ....
 *
 * \section section_tdm Table des matières
 * - \ref page_README
 * - \ref page_changelog
 * - \ref page_install
 * - \ref page_about
 * - \ref page_licence
 *
 */

/*! \page page_install Installation
 *
 * \todo rédiger le manuel d'installation
 *
 */

...
```

Remarque : il est possible de tester des *tags* si on les a ajoutés à la variable `ENABLED_SECTIONS = todo` du fichier `Doxyfile`

```
\if todo
- \ref todo
\endif
```

Lire :

- Présentation du format Markdown
- Pandoc
- Markdown avec VSCode

On pourra aussi utiliser Markdown :

```
\page page_README README

[TOC]

# Projet {#projet}

## Présentation {#presentation}


## Base de données {#bdd}

~~~ {.sql}

~~~

## Recette {#recette}


## Exemples {#exemples}

\snippet ./test.cpp Test

## Informations {#informations}

\author Thierry Vaira <<thierr.vaira@gmail.com>>
\date 2020
\version 0.1
\see https://svn.riouxsvn.com/projet
```

Exemples de fichier `Doxyfile` pour le projet : doxygen-projet-cpp.zip et doxygen-projet-java.zip

Exemple de rendu HTML : pour C++ ou pour Java

Exemple de rendu PDF : pour C++ ou pour Java

Exemple de rendu HTML : Revue 2 (projet BTS SNIR)

Règles de projet

Voici les règles à respecter lors des projets, vous devez documenter :

- chaque fichier (`@file`) au tout début du fichier
- chaque constante et/ou macro
- chaque type énuméré
- chaque structure
- chaque classe
- chaque attribut dans son fichier de déclaration
- chaque méthode dans son fichier de définition

Ensuite, on ajoutera les pages de documentation au format Markdown suivantes :

- une page principal
- une page README
- une page Changelog
- une page TODO

— une page À propos
— une page Licence

Exemples de fichier `Doxyfile` pour le projet : doxygen-projet-cpp.zip et doxygen-projet-java.zip

Exemple de rendu HTML : pour C++ ou pour Java

Exemple de rendu PDF : pour C++ ou pour Java

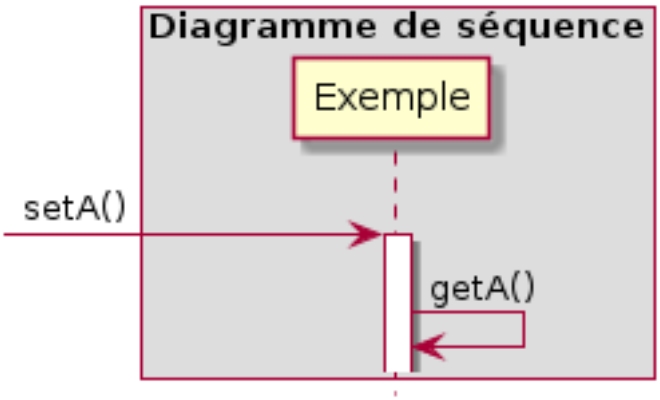
Exemple de rendu HTML : Revue 2 (projet BTS SNIR)

Extras

Diagrammes UML

Doxygen génère les diagrammes de classes et des graphes d'appels (`@callergraph` et `@callgraph`) en utilisant Graphviz.
Il est possible intégrer des diagrammes PlantUML dans une documention générée par Doxygen (activite-plantuml.pdf) :

```
/**
 * ...
 *
 *  \startuml
 *      hide footbox
 *      skinparam BoxPadding 50
 *      box "Diagramme de séquence"
 *      participant Exemple
 *      end box
 *      [-> Exemple: setA()
 *      Activate Exemple
 *      Exemple->Exemple: getA()
 *  \enduml
 */
```

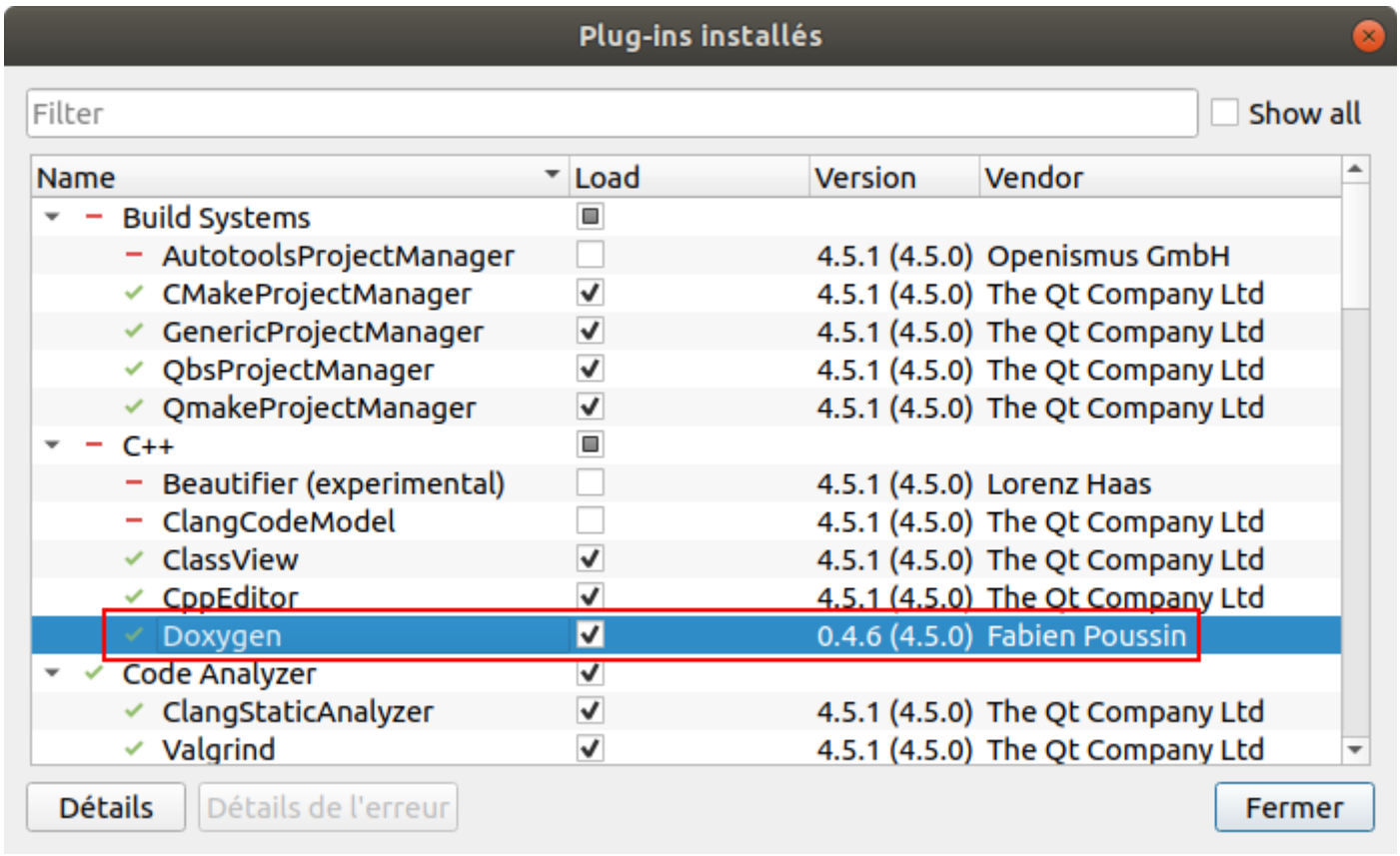


Un diagramme de séquence généré par PlantUML

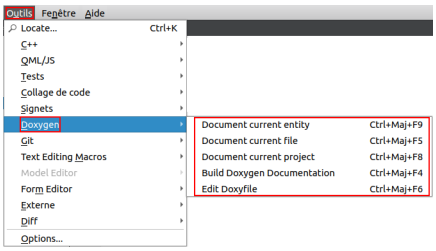
Il est aussi possible d'intégrer des diagrammes PlantUML avec Markdown sous VSCode : markdown-vscode.pdf.

Plugin Qt Creator

Il existe un plugin Doxygen pour Qt Creator qui permet d'intégrer dans l'EDI des fonctionnalités de documentation.
Aller dans **Aide** → **À propos des plug-ins...** :



Vous obtenez un nouveau sous-menu **Doxygen** dans le menu **Outils** :



D'autre part, vous aurez accès à la complétion pour les commandes Doxygen à partir de `@` ou `\` :

Pour les anciennes versions de Qt Creator :

Wiki : http://dev.kofee.org/projects/qtcreator-doxygen/wiki

Remarque : Le plugin est fourni sous forme binaire jusqu'à la version 3.1 de Qt Creator. Pour les versions supérieures, il vous faudra le recompiler à partir des sources.

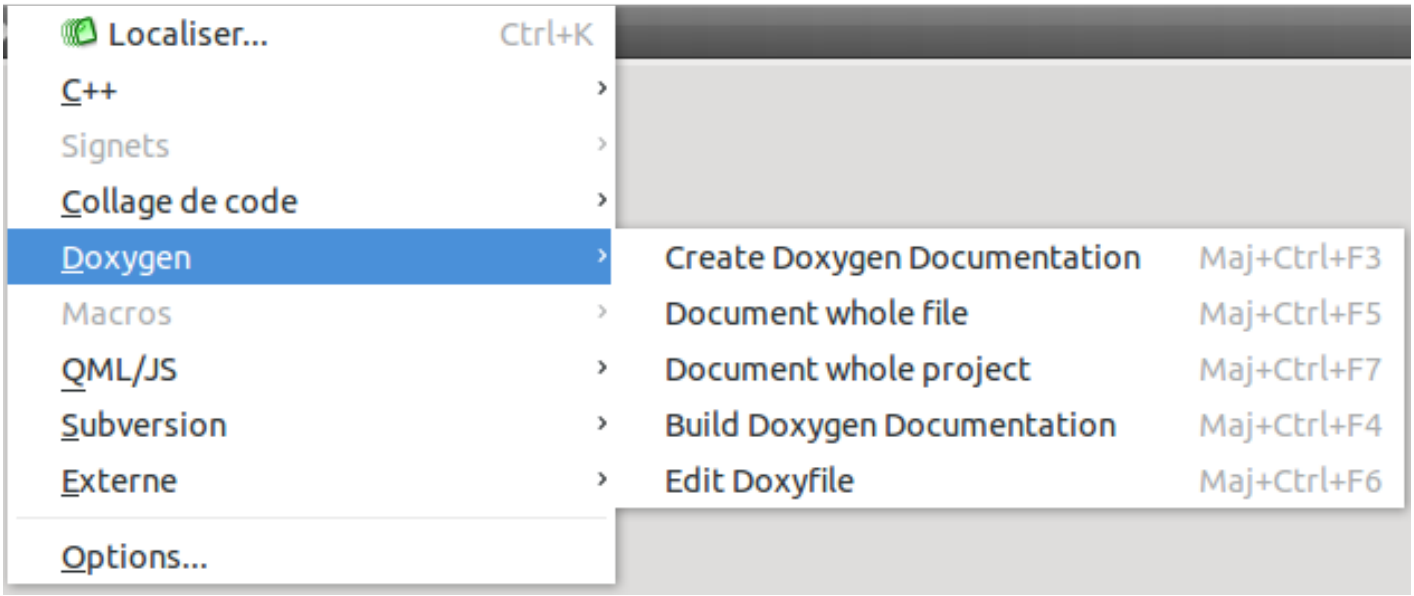
Pour la version Qt Creator 2.4.1 (64 bits) : qtcreator-doxygen-0.3.5-qtcreator-2.4.1-linux-x86_64.zip

Les plugins de Qt Creator sont installés dans `/usr/lib/x86_64-linux-gnu/qtcreator/plugins/`(pour une Ubuntu 12.04), donc :

```
$ sudo unzip qtcreator-doxygen-0.3.5-qtcreator-2.4.1-linux-x86_64.zip -d /usr/lib/x86_64-linux-gnu/qtcreator/
    plugins/
```

Il faut relancer Qt Creator.

Vous obtenez un nouveau sous-menu dans le menu Outils :



Utilisation :

- "Create Doxygen Documentation" : crée les en-têtes de documentation pour la ligne courante
- "Document whole file" : crée les en-têtes de documentation pour l'ensemble du fichier ouvert
- "Document whole project" : crée les en-têtes de documentation pour l'ensemble du projet actif
- "Build Doxygen Documentation" : fabrique la documentation (équivalent à Run dans doxywizard) du projet actif
- "Edit Doxyfile" : vous permet d'éditer votre fichier de configuration Doxyfile avec doxywizard

Intégration de Qt QML

Lien : github.com/agateau/doxyqml

Voir aussi

- http://axiomcafe.fr/tutoriel-documenter-un-code-avec-doxygen
- http://franckh.developpez.com/tutoriels/outils/doxygen/
- Une liste des commandes
- Doxygen et Graphviz

Retour au sommaire