简介 Doxygen

一．什么是 Doxygen?

Doxygen 是一个程序的文件产生工具，可将程序中的特定批注转换成为说明文件。通常我们在写程序时，或多或少都会写上批注，但是对于其它人而言，要直接探索程序里的批注，与打捞铁达尼号同样的辛苦。大部分有用的批注都是属于针对函式，类别等等的说明。所以，如果能依据程序本身的结构，将批注经过处理重新整理成为一个纯粹的参考手册，对于后面利用您的程序代码的人而言将会减少许多的负担。不过，反过来说，整理文件的工作对于您来说，就是沉重的负担。
Doxygen 就是在您写批注时，稍微按照一些它所制订的规则。接着，他就可以帮您产生出漂亮的文档了。
因此，Doxygen 的使用可分为两大部分。首先是特定格式的批注撰写，第二便是利用 Doxygen 的工具来产生文档。

目前 Doxygen 可处理的程序语言包含：

- C/C++
- Java
- IDL (Corba, Microsoft 及 KDE-DCOP 类型)

而可产生出来的文档格式有：

- HTML
- XML
- LaTeX
- RTF
- Unix Man Page

而其中还可衍生出不少其它格式。HTML 可以打包成 CHM 格式，而 LaTeX 可以透过一些工具产生出 PS 或是 PDF 文档。

二．安装 Doxygen

- 1.1 安装 Doxygen 1.7.4(Windows)
- 1.2 安装 graphviz 2.28.0(Windows)

graphviz 是一个由 AT&T 实验室启动的开源工具包，用于绘制 DOT 语言脚本描述的图形。Doxygen 使用 graphviz 自动生成类之间和文件之间的调用关系图，如不需要此功能可不安装该工具包。

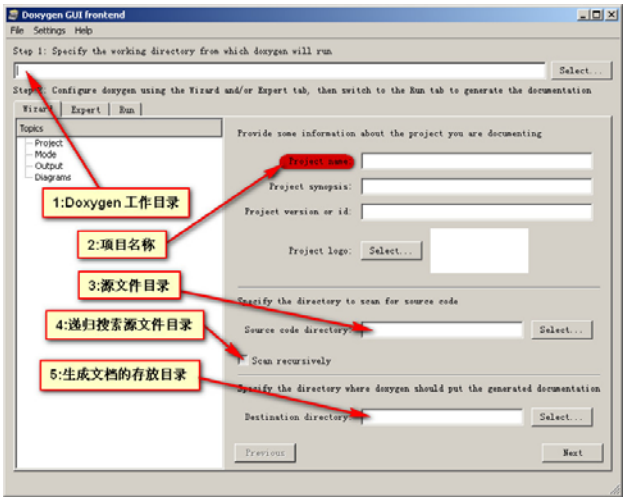
- 1.3 安装 Windows Help Workshop 1.32

Doxygen 使用这个工具可以生成 CHM 格式的文档。

三．Doxygen 的配置

Doxygen 产生文档可以分为三个步骤。一是在程序代码中加上符合 Doxygen 所定义批注格式。二是使用 Doxywizard 进行配置。三是使用 Doxygen 来产生批注文档。

Doxygen 1.7.4 主界面如下图 1 所示。

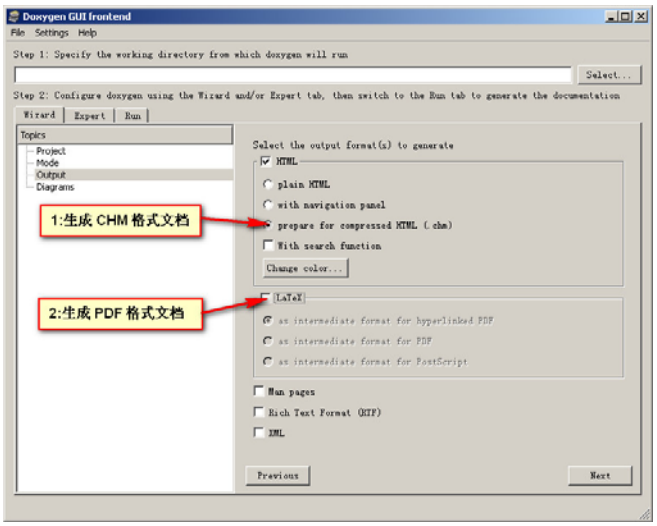


说明：1，Doxygen 工作目录，就是用来存放配置文件的目录。

2，递归搜索源文件目录需要选上。

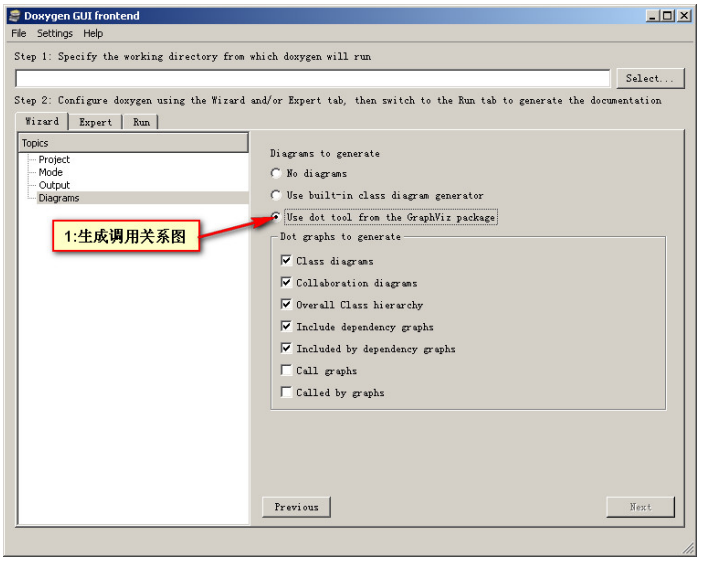
选择 wizard 标签下的 Output Topics

相关配置说明如下图 2 所示。



选择 wizard 标签下的 Diagrams Topics

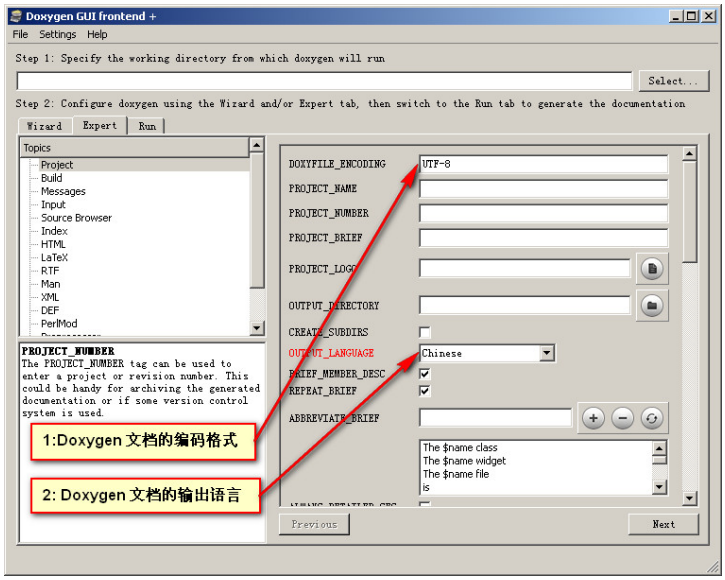
相关配置说明如下图 3 所示。



说明：如果选择这个选项之前需要先安装了 Graphviz 工具包。

选择 expert 标签下的 Project Topics

相关配置说明如下图 4 所示。



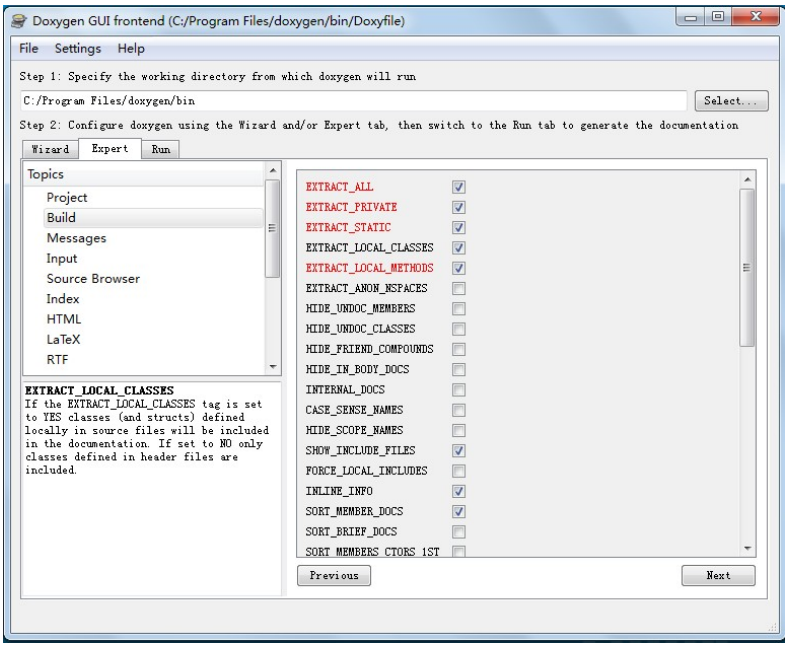
说明：编码格式，UTF-8 是首选。如果需要显示中文则选择 GB2313.

TAB_SIZE 主要是帮助文件中代码的缩进尺寸，譬如@code 和@endcode 段中代码的排版，建议设置成 4。

OPTIMIZE_OUTPUT_FOR_C 这个选项选择后，生成文档的一些描述性名称会发生变化，主要是符合 C 习惯。如果是纯 C 代码，建议选择。

SUBGROUPING 这个选项选择后，输出将会按类型分组。

选择 expert 标签下的 Build



Build 页面，这个页面是生成帮助信息中比较关键的配置页面：

EXTRACT_ALL 表示：输出所有的函数，但是 private 和 static 函数不属于其管制。

EXTRACT_PRIVATE 表示：输出 private 函数。

EXTRACT_STATIC 表示：输出 static 函数。同时还有几个 EXTRACT，相应查看文档即可。

HIDE_UNDOC_MEMBERS 表示：那些没有使用 doxygen 格式描述的文档（函数或类等）就不显示了。当然，如果 EXTRACT_ALL 被启用，那么这个标志其实是被忽略的。

INTERNAL_DOCS 主要指：是否输出注解中的@internal 部分。如果没有被启动，那么注解中所有的@internal 部分都将在目标帮助中不可见。

CASE_SENSE_NAMES 表示：是否关注大小写名称，注意，如果开启了，那么所有的名称都将被小写。对于 C/C++这种字母相关的语言来说，建议永远不要开启。

HIDE_SCOPE_NAMES 表示：域隐藏，建议永远不要开启。

SHOW_INCLUDE_FILES 表示：是否显示包含文件，如果开启，帮助中会专门生成一个页面，里面包含所有包含文件的列表。

INLINE_INFO ：如果开启，那么在帮助文档中，inline 函数前面会有一个 inline 修饰词来标明。

SORT_MEMBER_DOCS ：如果开启，那么在帮助文档列表显示的时候，函数名称会排序，否则按照解释的顺序显示。

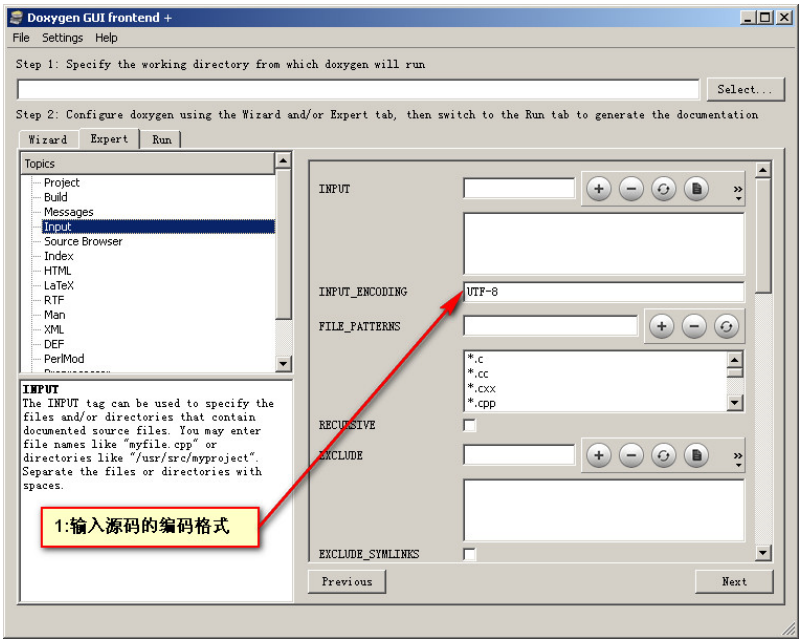
GENERATE_TODOLIST ：是否生成 TODOLIST 页面，如果开启，那么包含在@todo 注解中的内容将会单独生成并显示在一个页面中，其他的 GENERATE 选项同。

SHOW_USED_FILES ：是否在函数或类等的帮助中，最下面显示函数或类的来源文件。

SHOW_FILES ：是否显示文件列表页面，如果开启，那么帮助中会存在一个一个文件列表索引页面。

选择 expert 标签下的 Input Topics

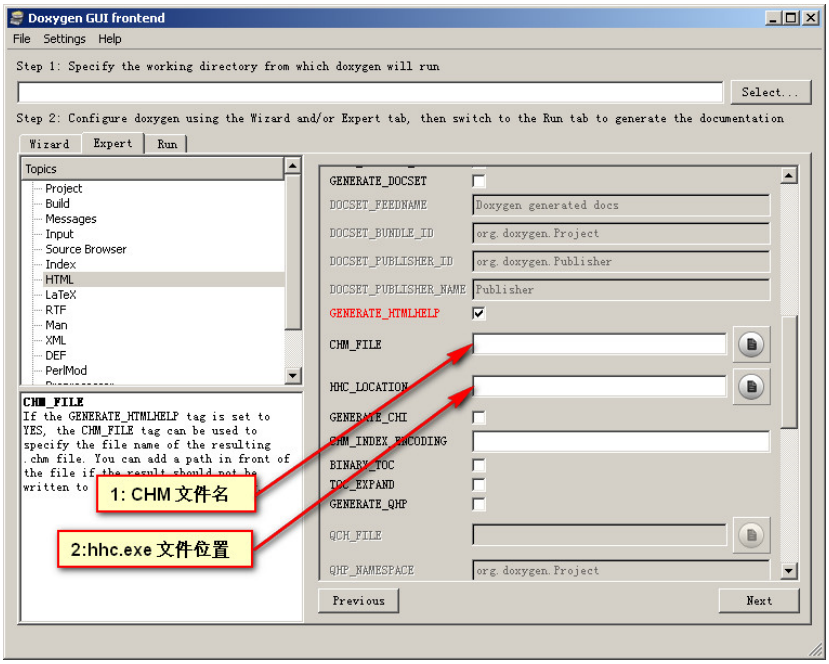
相关配置说明如下图 5 所示。



说明：输入的源文件的编码，要与源文件的编码格式相同。如果源文件不是 UTF-8 编码最好转一下。

选择 expert 标签下的 HTML Topics

相关配置说明如下图 6 所示。



说明：1，CHM_FILE 文件名需要加上后缀（xx.chm）。

2，如果在 Wizard 的 Output Topics 中选择了 prepare for compressed HTML (.chm)选项，此处就会要求选择 hhc.exe 程序的位置。在 windows help workshop 安装目录下可以找到 hhc.exe。

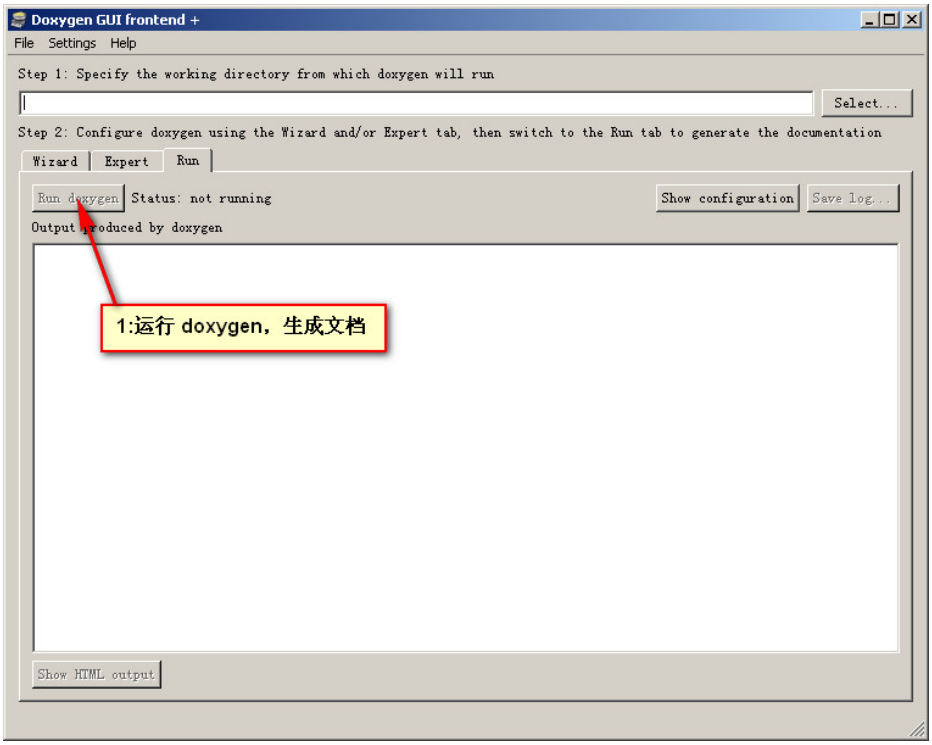
3，为了解决 DoxyGen 生成的 CHM 文件的左边树目录的中文变成了乱码，CHM_INDEX_ENCODING 中输入 GB2312 即可。

4，GENERATE_CHI 表示索引文件是否单独输出，建议关闭。否则每次生成两个文件，比较麻烦。

5，TOC_EXPAND 表示是否在索引中列举成员名称以及分组（譬如函数，枚举）名称。

选择 Run 标签

相关配置说明如下图 7 所示。



点击 Run doxygen 按钮， Doxygen 就会从源代码中抓取符合规范的注释生成你定制的格式的文档。

四．撰写正确格式的批注

并非所有程序代码中的批注都会被 Doxygen 所处理。您必需依照正确的格式撰写。原则上，Doxygen 仅处理与程序结构相关的批注，如 Function，Class ，档案的批注等。对于 Function 内部的批注则不做处理。Doxygen 可处理下面几种类型的批注。

JavaDoc 类型：

```
/**
  * ... 批注 ...
  */
```

Qt 类型：

```
/*!
  * ... 批注 ...
  */
```

单行型式的批注：

```
/// ... 批注 ...
或
//! ... 批注 ...
```

要使用哪种型态完全看自己的喜好。以笔者自己来说，大范围的注解我会使用 JavaDoc 型的。单行的批注则使用"///" 的类型。

此外，由于 Doxygen 对于批注是视为在解释后面的程序代码。也就是说，任何一个批注都是在说明其后的程序代码。如果要批注前面的程式码则需用下面格式的批注符号。

```
/*!< ... 批注 ... */
/**< ... 批注 ... */
//!< ... 批注 ...
///< ... 批注 ...
```

上面这个方式并不适用于任何地方，只能用在 class 的 member 或是 function 的参数上。

举例来说，若我们有下面这样的 class。

```
class MyClass {
        public:
                int member1 ;
                int member2:
                void member_function();
};
```

加上批注后，就变成这样：

```
/**
  * 我的自订类别说明 ...
  */
class MyClass {
        public:
                int member1 ; ///< 第一个 member 说明 ...
                int member2:   ///< 第二个 member 说明 ...
                int member_function(int a, int b);
};

/**
  * 自订类别的 member_funtion 说明 ...
```

```
  *
  * @param a 参数 a 的说明
  * @param b 参数 b 的说明
  *
  * @return 传回 a+b。
  */
int MyClass::member_function( int a, int b )
{
        return a+b ;
}
```

当您使用 Doxygen 产生说明文档时，Doxygen 会帮您 parsing 您的程式码。并且依据程序结构建立对应的文件。然后再将您的批注，依据其位置套入于正确的地方。您可能已经注意到，除了一般文字说明外，还有一些其它特别的指令，像是@param 及@return 等。这正是 Doxygen 另外一个重要的部分，因为一个类别或是函式其实都有固定几个要说明的部分。为了让 Doxygen 能够判断，所有我们就必需使用这些指令，来告诉 Doxygen 后面的批注是在说明什么东西。Doxygen 在处理时，就会帮您把这些部分做特别的处理或是排版。甚至是制作参考连结。

首先，我们先说明在 Doxygen 中对于类别或是函数批注的一个特定格式。

```
/**
  * class 或 function 的简易说明...
  *
  * class 或 function 的详细说明...
  * ...
  */
```

上面这个例子要说的是，在 Doxygen 处理一个 class 或是 function 注解时，会先判断第一行为简易说明。这个简易说明将一直到空一行的出现。或是遇到第一个"." 为止。之后的批注将会被视为详细说明。两者的差异在于 Doxygen 在某些地方只会显示简易说明，而不显示详细说明。如：class 或 function 的列表。

另一种比较清楚的方式是指定@brief 的指令。这将会明确的告诉 Doxygen，何者是简易说明。例如：

```
/**
  * @brief class 或 function 的简易说明...
  *
  * class 或 function 的详细说明...
```

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

除了这个 class 及 function 外，Doxygen 也可针对档案做说明，条件是该批注需置于档案的前面。主要也是利用一些指令，通常这部分注解都会放在档案的开始地方。如：

```
/*! \file myfile.h
        \brief 档案简易说明

        详细说明.

        \author 作者信息
*/
```

如您所见，档案批注约略格式如上，请别被"\" 所搞混。其实，"\" 与"@" 都是一样的，都是告诉 Doxygen 后面是一个指令。两种在 Doxygen 都可使用。笔者自己比较偏好使用"@"。

接着我们来针对一些常用的指令做说明：

@file	档案的批注说明。
@author	作者的信息
@brief	用于 class 或 function 的批注中，后面为 class 或 function 的简易说明。
@param	格式为 @param arg_name 参数说明 主要用于函式说明中，后面接参数的名字，然后再接关于该参数的说明。
@return	后面接函数传回值的说明。用于 function 的批注中。说明该函数的传回值。
@retval	格式为 @retval value 传回值说明 主要用于函式说明中，说明特定传回值的意义。所以后面要先接一个传回值。然后在放该传回值的说明。

Doxygen 所支持的指令很多，有些甚至是关于输出排版的控制。您可从 Doxygen 的使用说明中找到详尽的说明。

下面我们准备一组 example.h 及 example.cpp 来说明 Doxygen 批注的使用方式：

example.h:

```
/**
  * @file 本范例的 include 档案。
  *
  * 这个档案只定义 example 这个 class。
  *
  * @author garylee@localhost
  */

#define EXAMPLE_OK   0     ///< 定义 EXAMPLE_OK 的宏为 0。

/**
  * @brief Example class 的简易说明
  *
  * 本范例说明 Example class。
  * 这是一个极为简单的范例。
  *
  */
class Example {
        private:
                int var1 ; ///< 这是一个 private 的变数
        public:
                int var2 ; ///< 这是一个 public 的变数成员。
                int var3 ; ///< 这是另一个 public 的变数成员。
                void ExFunc1(void);
                int ExFunc2(int a, char b);
                char *ExFunc3(char *c) ;
};
```

example.cpp:

```
/**
  * @file 本范例的程序代码档案。
  *
  * 这个档案用来定义 example 这个 class 的
  * member function。
  *
  * @author garylee@localhost
  */

/**
```

```
  * @brief ExFunc1 的简易说明
  *
  * ExFunc1 没有任何参数及传回值。
  */
void Example::ExFunc1(void)
{
        // empty funcion.
}

/**
  * @brief ExFunc2 的简易说明
  *
  * ExFunc3()传回两个参数相加的值。
  *
  * @param a 用来相加的参数。
  * @param b 用来相加的参数。
  * @return 传回两个参数相加的结果。
  */
int ExFunc2(int a, char b)
{
        return (a+b);
}

/**
  * @brief ExFunc3 的简易说明
  *
  * ExFunc3()只传回参数输入的指标。
  *
  * @param c 传进的字符指针。
  * @retval NULL 空字符串。
  * @retval !NULL 非空字符串。
  */
char * ExFunc2(char * c)
{
        return c;
}
```