Doxygen v1.63 中文手册

Anan

---------- **【 声 明 】** ----------

1. 此为免费文档，请勿用于商业用途。
2. 对于本手册的任何问题，请到各大专业论坛发问，我相信你会得到很多人的帮助。

---------- **【 约 定 】** ----------

1. 对于意思不明确难以判断的句子，在给出的译文之后紧跟红色标记的原文 to get red origin text after translated text 。
2. 对于省略的部分，即带有（略）字提示的部分，依旧会给出原文，以保证文档的完整性。
3. doxygen 配置文件中的选项标记一律使用蓝色提示。
4. 文中的内容备注一律使用橙色提示。
5. 对于文中所有的标题均用**黑色粗体**表示。
6. 文中的符号一律使用“”双引号标注，除去特别加入\“\”反斜杠表示转义之外，双引号无意义。
7. 文中的章节名称一律使用“”双引号**粗体**标注，所以与符号不存在冲突。
8. 原文的错误一律使用绿色提示，并在译文之后给出，错处用下划线标注。
9. 文中所有的备注，使用“**”开头以及深红色进行标识。

---------- **【 问题】** ----------

---------- 【 正文 】 ----------

此手册被分为三个部分，每个部分都包含若干章节。

第一部分是用户指南

1. **安装**，讨论如何在你的平台中下载，编译和安装 doxygen。
2. **开始**，告诉你如何快速生成第一个 doxygen 文档。
3. **代码文档化**，演示代码被文档化的若干方法。
4. **列表**，显示创建列表的若干方法。
5. **组合**，显示如何将一些工作组合在一起。
6. **包含公式**，显示如何在文档中插入公式。
7. **图形和图表**，描述 doxygen 生成图表和图形。
8. **预处理**，解析 doxygen 如何处理宏定义。
9. **生成自动链接**，显示如何在文档中放置文件，类，成员的链接。
10. **输出格式**，显示如何生成 doxygen 支持的多种输出格式。
11. **搜索**，显示在 HTML 文档中进行搜索的若干方法。
12. **自定义输出**，解析如何在 doxygen 中生成自定义的输出。
13. **自定义命令**，显示如何在你的注释中定义和使用自定义的命令。
14. **连接到外部文档**，解析如何让 doxygen 创建与外部已生成文档的链接。
15. **FAQ**，对频繁出现的提问给出的回答。
16. **排错**，当出现问题时，它会告诉你该做些什么。

第二部分是参考指南

1. **特点**，展示 doxygen 的功能概览。
2. **doxygen 历史**，告知 doxygen 在开发过程中的变化，以及它一直在进行的内容。
3. **doxygen 用法**，告知如何使用 doxygen。
4. **doxytag 用法**，告知如何使用 doxytag。
5. **doxywizard 用法**，告知如何使用 doxywizard。
6. **installdox 用法**，告知如果你使用 tag 文件，在 doxygen 中如何使用生成的 installdox 脚本。
7. **配置**，告知如何调整 doxygen，以便生成你需要的文档。
8. **特殊命令字**，告知在文档中能使用的特殊命令的概览。
9. **HTML 命令字**，告知在文档中能使用的 HTML 命令的概览。
10. **XML 命令字**，告知在文档中能使用的 C#风格的 XML 命令的概览。

第三部分为 doxygen 开发者提供的一些信息（略）

1. **doxygen 内部构造**，给出 doxygen 内部结构的整体概览。
2. **Perl 模块的输出格式**，显示如何使用 PerlMod 的输出。
3. **国际化**，解析如何添加新语种的输出支持。

安　　装

如果你还没有 doxygen 程序包，首先可以去 http://www.doxygen.org/download.html 页面获得最新的版本。

本章节包含以下几部分：

Unix 源码包编译
Unix 二进制包安装
Unix 已知的编译问题
Windows 源码包编译
Windows 二进制包安装
Doxygen 开发工具（略）

Unix 源码包编译

如果你已下载了源码包，你需要按照以下的步骤来创建可执行文件：

1. 需要 GNU 工具软件，flex，bison，GNU make，strip
2. 为了生成 Makefile，你需要安装 Perl。
3. 配置脚本已假定标准的 Unix 工具，例如 sed，date，find，uname，mv，cp，cat，echo，tr，cd，rm 都是可用的。

想获得 doxygen 的所有功能，以下的辅助工具必须安装。

1. QT 4.3 或者更高版本，用于创建 GUI 后端 doxywizard。
2. 一个 OCR 种类的发布程序，比如 teTex 1.0，用于生成 LaTex，Postscript，PDF 的输出。
3. Graph visualization toolkit 1.8 或者更高版本，doxygen 中的 graphs, graphical inheritance graphs, collaboration graphs 与 graphviz 都存在依赖关系，如果你准备重新编译 graphviz，确认系统是否支持 freetype，否则 graphs 无法渲染特殊文本的标记。
4. 如果需要显示公式，并且你不想使用 pdflatex，那么 ghostscript 也是必须的，你可以在 www.ghostscript.com 找到它。
5. 为了生成 doxygen 自身的文档，需要 Python，你可以在 www.python.org 找到它。

执行以下的步骤进行编译：

1. 解压程序包
```
gunzip doxygen-$VERSION.src.tar.gz    # uncompress the archive
tar xf doxygen-$VERSION.src.tar       # unpack it
```

2. 运行配置脚本
```
sh ./configure
```

此脚本尝试检测你所使用的平台，make（必须是 GNU make），以及 Perl，并且脚本会输出检测结果。
你可以运行以下命令来配置，使用检测出的平台和编译器来替换掉命令中的 platform-type。
```
configure  -platform platform-type
```

比对一下程序包中 PLATFORMS 文件所列出的平台与命令中的 platform-type。

如果系统中存在 QT 4.3 或者更高的版本，并需要创建 GUI 后端，要运行配置脚本的--with-doxywizard 选项，
```
configure --with-doxywizard
```

其他的配置选项的概览可以使用，
```
configure --help
```

3. 运行 make 编译程序
```
make
```

如果程序编译无任何问题，在系统的 bin 目录下将会有三个二进制可执行文件，doxygen，doxytag，doxywizard。

4. 生成用户手册
```
make docs
```

让 doxygen 生成 HTML 文档，在系统中的 HTML 目录下将包含此文档（目录中的 index.html 文件将指向一个 HTML 浏览器），此时你将需要 python 的支持。

5. 生成一个 PDF 版本的用户手册（此时你需要 pdflatex，makeindex，egrep 的支持）
```
make pdf
```

doxygen_manual.pdf 将放置在系统的 latex 目录下，可以使用 acrobat reader 查看和打印。

6. 安装 doxygen
```
make install
```

Unix 二进制包安装

如果你下载的是一个 Unix 的二进制发行版本，输入：

```
./configure
Make install
```

可执行的二进制文件将被安装到<prefix>/bin 目录下，使用 make install_doc 安装 doxygen 的文档和例子，安装目录为<docdir>/doxygen。

<prefix>默认路径为/usr/local, 可使用配置脚本的--prefix 选项来修改安装的默认路径。
<docdir>默认路径为<prefix>/share/doc/packages,可是使用配置脚本的--docdir 选项来修改默认路径。

当然你也可以直接复制 doxygen 二进制文件，到/bin 或者 PATH 变量中能被检索到 bin 目录下，这一点对使用 doxygen 很重要。

注意：
你需要 GNU 安装工具来完成上述的工作（这只是所需支持程序包的一部分），而其他的安装支持工具有可能被放置到错误的目录下，请确认！如果你使用的是一个 RPM 或者 DEP 软件包，请遵照这些安装包的标准安装过程进行。

Unix 已知的编译问题

Qt 的问题

一些系统中 Qt 的头文件和库文件不在 QTDIR 所指向的目录下（比如 Red Hat 6.0，头文件路径/usr/include/qt, 库文件路径/usr/lib)。

解决方法：

进入 doxygen 的根目录，

```
mkdir qt
cd qt
ln -s your-qt-include-dir-here include
ln -s your-qt-lib-dir-here lib
export QTDIR=$PWD
```

如果你使用 C 风格的 shell，可用 setenv QTDIR $PWD 来替换 export 那一行。

Bison 问题

Bison 1.31~1.34 版本都存在一个编译错误：
ce_parse.cpp:348: member `class CPPValue yyalloc::yyvs' with constructor not allowed in union
这个问题已经 1.35 版本得到解决（1.31 之前的版本也不存在问题）。

Latex 问题

a4wide.sty 文件不是在所有的系统都有效，如果你的系统无法使用，请选择配置文件中其他的页面类型（查看配置文件的 PAPER_TYPE 标记下的相关信息）。

HP-UX & Digital Unix 问题

如果你使用 aCC 在 HP-UX 中编译，会得到这个错误：

```
/opt/aCC/lbin/ld: Unsatisfied symbols:
alloca (code)
```

你可根据 Anke Selig 的提示，用 #include <alloca.h> 替换掉 ce_parse.cpp 文件的以下部分，

```
extern "C" {
    void *alloca (unsigned int);
};
```

如果没有任何帮助，尝试删除 ce_parse.cpp 文件，重新安装 bison 软件包。

如果在 Digital Unix 下编译，相同的问题可根据 Barnard Schmallhof 提示，在 ce_parse.cpp 文件中，使用

```
#else /* not GNU C.  */
   #if (!defined (__STDC__) && defined (sparc)) || defined (__sparc__) \
       || defined (__sparc) || defined (__sgi) || defined (__osf__)
   #include <alloca.h>
```

替换掉以下内容，

```
#else /* not GNU C.  */
   #if (!defined (__STDC__) && defined (sparc)) || defined (__sparc__) \
       || defined (__sparc) || defined (__sgi)
   #include <alloca.h>
```

还需要修正 bison 的一个问题，以下是 bison.simple 文件的 patch（由 Andre Johansen 提供）

```
--- bison.simple~      Tue Nov 18 11:45:53 1997
+++ bison.simple       Mon Jan 26 15:10:26 1998
@@ -27,7 +27,7 @@
 #ifdef __GNUC__
 #define alloca __builtin_alloca
```

```
 #else /* not GNU C.  */
-#if (!defined (__STDC__) && defined (sparc)) || defined (__sparc__) \
    || defined (__sparc) || defined (__sgi)
+#if (!defined (__STDC__) && defined (sparc)) || defined (__sparc__) \
    || defined (__sparc) || defined (__sgi) || defined (__alpha)
 #include <alloca.h>
 #else /* not sparc */
 #if defined (MSDOS) && !defined (__TURBOC__)
```

此 patch 能在 doxygen 创建过程生成 scanner.cpp 文件。

Sun 编译问题

如果使用 Sun C++ WorkShop 6 编译器，出现 doxygen 无法正常工作的问题，我自己也不能确定真正的原因，我还没有获得操控一台 Solaris 主机和编译器的途径。但是使用 GNU 编译 doxygen 是可以工作的，并且安装 Sun patch 111679-13 可以修正这个问题。

当配置时添加--static 选项，可以得到：

```
Undefined                       first referenced
 symbol                             in file
dlclose                             /usr/lib/libc.a(nss_deffinder.o)
dlsym                               /usr/lib/libc.a(nss_deffinder.o)
dlopen                              /usr/lib/libc.a(nss_deffinder.o)
```

手动添加 -Bdynamic 在 Makefile.doxygen 和 Makefile.doxytag 文件的 target rule 之后，如下：

```
$(TARGET): $(OBJECTS) $(OBJMOC)
    $(LINK) $(LFLAGS) -o $(TARGET) $(OBJECTS) $(OBJMOC) $(LIBS) -Bdynamic
```

GCC 编译问题

老版本的 GNU 编译器存在常量字符串的字符编码大于 127 的问题，并且在编译 translator_xx.h 文件时会出错，如果你只准备使用英文字符，配置脚本中使用--english-only 选项可以规避以上的问题。

在一些平台（例如 OpenBSD）中，某些 gcc 版本在使用-O2 参数编译文件（比如 config.cpp）期间会产生内存泄漏，配置脚本中使用--debug 选项或者删除掉 Makefile 中出错文件的-O2 编译参数，来规避内存泄漏。

因为存在 bug，Gcc 2.95 之前版本都可以终止二进制文件的执行。Gcc versions before 2.95 may produce broken binaries due to bugs in these compilers.

Dot 问题

因为图元 image maps 生成方法的改变，老版本的 doxygen(<=1.2.17)无法与新版本的 graphviz(>=1.8.8)一同正

常工作。这种不兼容的结果导致HTML中生成graphs无法正常点击。推荐使用doxygen 1.3版本匹配graphviz 1.8.10或者更高的版本。推荐使用 doxygen 1.4.7版本匹配 graphviz 2.8或者更高的版本，以避免字体的问题。

Red Hat 9.0 问题

如果你运行make之后，得到以下的错误，
  tmake error: qtools.pro:70: Syntax error

在运行make之前，输入 export LANG=

Windows 源码包编译

从1.5.0版本开始支持Visual Studio 2005创建程序，当然免费的“专业”版本开发环境也能用来编译doxygen。或者安装Cygwin或MinGW采用Unix的方式进行编译。

编译doxygen之前需要下载和安装visual studio的c++编译器，自从微软开始明显地诱惑所有人吸食他们的.net毒品之后，他们使得运用专业版本时的难度增加了，所以你需要手动完成一些步骤，来定制一个合适的工作环境，编译一个原始的win32应用程序，比如doxygen。

http://msdn2.microsoft.com/en-gb/express/aa700755.aspx
配置vs2005的win32环境

下一步安装unxutils，http://sourceforge.net/projects/unxutils ，此软件包囊括了flex和bison工具，它们在编译处理时是必须的，如果你使用的是一个doxygen的CVS，在官方的发行版本中包含有此软件包。下载zip文件并将其放置到，例如C:\tools\unxutils目录下.

现在你需要增加和调整系统的环境变量，

1. 增加C:\tools\unxutils\usr\local\wbin目录到PATH中
2. 添加变量BISON_SIMPLE，为其设定C:\tools\unxutils\usr\local\share\bison.simple目录。

下载doxygen的源码包，放置到任意位置，比如C:\tools

打开命令行界面，输入

```
cd c:\tools
gunzip doxygen-x.y.z.src.tar.gz
tar xvf doxygen-x.y.z.src.tar
```

解压源码包，现在环境被安装好了，可以创建doxygen和doxytag。

进入doxygen-x.y.z目录，将会找到一个winbuild目录中包含一个名为Doxygen.sln文件，在visual studio打

开此文件，使用鼠标右键点击，解决方案浏览器中的项目菜单，构建 doxygen 和 doxytag 的发行版本和调试版本，或选择创建。

注意当前编译 doxywizard 需要 QT 4，http://www.trolltech.com/products/qt/qt3 ，如果你没有一个商业许可证，可以使用一个开源版本来创建 doxywizard
http://qtwin.sourceforge.net/qt3-win32/compile-msvc-2005.php
但是我自己没有尝试过。

也可以阅读下一章了解你安装和运行 doxygen 所需要的附加工具，和其他的一些特性。

Windows 二进制包安装

Doxygen 的 Windows 二进制包是一个自动安装文件，所以安装非常简单，只需按照对话框提示即可。

安装完成之后，推荐下载和安装 GraphViz（2.8 或者更高的推荐版本）。Doxygen 使用 graphviz 中的 dot 工具来渲染出效果更好的图表，查看一下配置文件的 HAVE_DOT 的选项。

如果在配置文件中，你需要处理压缩的 HTML 文件（需要查看 GENERATE_HTMLHELP），那么你还需要微软的 HTML help workshop 软件，可以从
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/htmlhelp/html/vsconHH1Start.asp
下载。

如果在配置文件中，你需要处理 QT 压缩的帮助文件（需要查看 QHG_LOCATION），那么你还需要 QT 程序包中 qhelpgenerator 软件，你可从 http://trolltech.com/downloads/下载。

为了生成 PDF 文件输出或使用科学公式，你将需要安装 LaTex 和 Ghostscript 软件。

LaTex 有许多发行版本存在，与 doxygen 协同工作的主流版本有 MikTex 和 XemTex。

Ghostscript 可从 Sourceforge 下载到。
http://sourceforge.net/projects/ghostscript/

安装 LaTex 和 Ghostscript 之后，你需要确认一下 latex.exe，pdflatex.exe, gswin32c.exe 的路径是否出现在 PATH 中，如果你无法确认，可以在“开始-运行”中直接输入程序名确认它是否能访问，如果不成功，要进行修正直到它可以工作为止。

Tools used to develop doxygen

Doxygen was developed and tested under Linux & MacOSX using the following open-source tools:

- GCC version 3.3.6 (Linux) and 4.0.1 (MacOSX)
- GNU flex version 2.5.33 (Linux) and 2.5.4 (MacOSX)
- GNU bison version 1.75
- GNU make version 3.80
- Perl version 5.8.1
- VIM version 6.2
- Firefox 1.5
- Trolltech's tmake version 1.3 (included in the distribution)
- teTeX version 2.0.2
- CVS 1.12.12

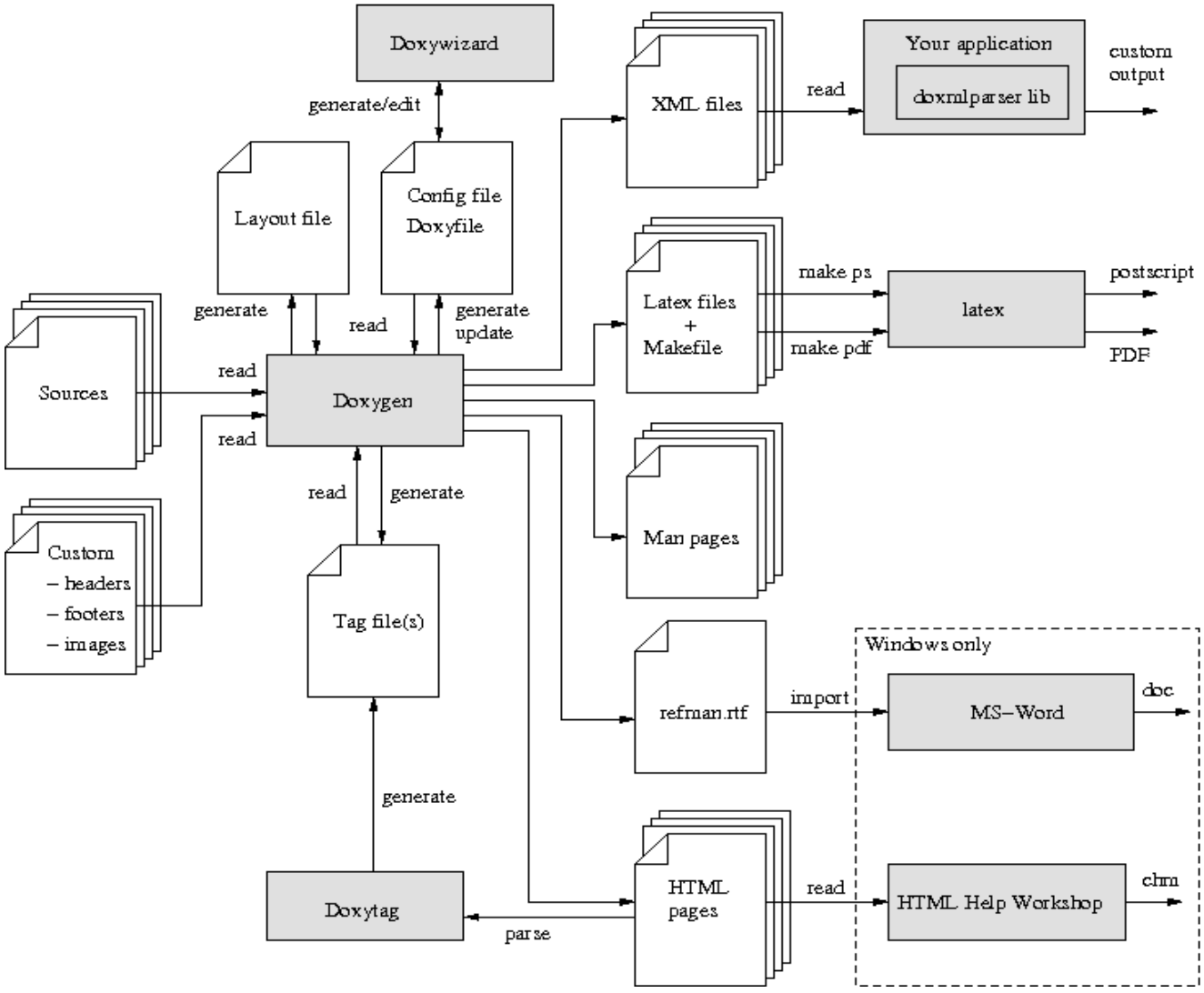
开　　始

doxygen 是这个软件包的主程序，用来解析源码包和生成文档。查阅“**doxygen 用法**”那一章获得更多的信息。

在你未获得目标源码包的情况下，却想建立与外部文档（由 doxygen 生成的）的引用，只需要 doxytag，查阅“**doxytag 用法**”那一章获得更多的信息。

可以选择是否使用 doxywizard 这个图形后端，它是用来启动 doxygen 的图形环境和编辑配置文件。在 Mac OS X 系统下点击 doxygen 应用图标将会启动 doxywizard。

以下的图解揭示出工具和信息流之间的相互关系（它看上去很复杂，是为了完整展现 doxygen 的全部功能）；



第一步：创建一个配置文件

Doxygen 使用一个配置文件来包含它的所有设定，每一个项目可以生成自己的配置文件。一个项目可以只含有一个单独的源文件，但是也会有一个能递归扫描的代码树。

简单创建配置文件的方法是使用 doxygen 提供给你的模板。在命令行中使用：
doxygen -g <config-file>

config-file 即为配置文件的名称，如果忽略文件名，将会有一个默认 doxyfile 的配置文件产生。假如 config-file 中设定的文件名已经存在，那么 doxygen 在生成配置模板之前，将定名一个 config-file.back 文件。如果你使用“-”(比如减号)来替代文件名，那么 doxygen 会尝试从标准输入（stdin）中读取配置文件，这一点对于脚本很有用。

配置文件的格式很简单，如同一个简单的 Makefile，它包含一定数量的标记符（tags）：
TAGNAME = VALUE or
TAGNAME = VALUE1 VALUE2 ...

你可以在一个生成的模板配置文件中，对大多数的 tags 使用他们的默认值。查阅“配置”章节获得关于配置文件的更多细节。

如果你不希望在一个文本编辑器中编辑配置文件，那你可以看看 doxywizard，这个用于创建，读取和写入 doxygen 配置文件的 GUI 后端，也允许通过对话框来选择你的配置选项。

对于一个包含少量 C/C++源文件和头文件的小项目，你可以使用空的 INPUT 标记（tag），它将使 doxygen 搜索当前目录下所有的源文件。

假如你有一个包含源码目录或源码树的大型项目，你需要分配 INPUT 标记给根目录或者所有目录，增加一个或者多个文件格式给 FILE_PATTERNS 标记（例如*.cpp，*.h），并且只有与 FILE_PATTERNS 中格式匹配的文件才会被解析（假定 FILE_PATTERNS 作为一个正在使用的扩展名列表的补充，否则它将无意义）。为了能对一个源码树进行递归解析，你必须设置 RECURSIVE 标记为 yes。为进一步对源码文件的解析方式做出调整，可以使用 EXCLUDE 和 EXCLUDE_PATTERNS 标记。例如忽略源码树中所有的 test 目录，可用这种方法：

```
EXCLUDE_PATTERNS = */test/*
```

Doxygen 通过查看文件的扩展名来确定如何解析该文件，如果一个文件的扩展名为 .idl 或 .odl，那么它将被看成一个 IDL 文件，.java 看成一个 java 文件，.cs 看成 C#文件，.py 则可选用 Python 解析器，.php，.php4，.inc，.phtml 都被看成是 PHP 的源文件，C/C++文件可被解析，.m 文件将被看成 Objective-C 的源文件。

如果你开始将 doxygen 用于一个已存在的项目（之前并未使用 doxygen 处理过的项目），你将能得到项目的构造

以及文档应如何展开的建议。想做到这些，你必须在配置文件设置 EXTRACT_ALL 标记为 YES，然后 doxygen 就可在源码被文档化的过程中自动完成所有的事情。
请注意一条重要的警告，只要设置 EXTRACT_ALL 标记为 YES，那么未公开的成员将不会出现在文档中。

为分析软件包的某一部分，在源代码中建立定义与文档的交叉引用是很有帮助的，To analyse an existing piece of software it is useful to cross-reference a (documented) entity with its definition in the source files，如果你设置 SOURCE_BROWSER 标记为 YES，doxygen 将生成交叉引用，INLINE_SOURCES 标记为 YES，在文档中将会直接包含源码文件（如此就很容易能够预览源代码）。

第二步：运行 doxygen

输入以下命令生成记录：
doxygen <config-file>

根据你的设置 doxygen 在输出目录中将生成 html，rtf，latex，xml 或者 man 目录，目录名称已经表明这些目录将包含 HTML，RTF，LaTex，XML 以及 Unix-Man 格式的记录。

默认的输出目录即为运行 doxygen 的目录。更改输出的根目录可以修改 OUTPUT_DIRECTORY 标记，输出目录中特定格式的输出目录也可以通过修改配置文件中的 HTML_OUTPUT, RTF_OUTPUT, LATEX_OUTPUT, XML_OUTPUT, MAN_OUTPUT 标记进行选择，假设输出目录不存在，doxygen 将为你创建一个（但 doxygen 不用尝试递归创建一个完整路径，如同 mkdir -p 所做的那样）。

HTML 输出

生成的 HTML 文档能够通过 html 目录中，index.html 文件所指向的 HTML 浏览器进行查看。所使用的浏览器最好能够支持 CSS（cascading style sheets）（我用 Mozilla, Safari, Konqueror，有时用到 IE6 来测试生成的输出文档）。

HTML 文档的一些特性（比如 GENERATE_TREEVIEW 或者搜索引擎），需要 DHTML 和 JavaScript 的支持。

LaTex 输出

想生成 LaTex 的文档，首先要运行 LaTex 的编译器进行编译（我是用最新版本的 teTex）。Doxygen 编写了一个 Makefile 放在 latex 目录下，它使得生成文档的编译处理变得很简单。

Makefile 中的目标和内容依赖与 **USE_PDFLATEX** 标记的设定。如果它无效（设置为 NO），在 latex 目录下输入 make 将生成一个 refman.dvi 文件，此文件可用 xdvi 查看或者输入 make ps 转换成一个 refman.ps 文件（此处需要 dvips 软件包）。

在一个物理页面放置 2 个文本页面，可用 make ps_2on1，PostScript 文件可发送给一个 PostScript 打印机。如果你没有一个 PostScript 打印机，可使用 Ghostscript 转换成你的打印机支持的格式。

如果你已安装了 Ghostscript 解释器也可以转换成 PDF，只需输入 make pdf 或者 make pdf_2on1 。

为获得 PDF 最好的输出效果，你需要设置 **PDF_HYPERLINKS** 和 **USE_PDFLATEX** 标记为 YES，这种情况下 Makefile 只包含一个目标，直接创建 refman.pdf 。

RTF 输出

Doxygen 将 RTF 输出组合到一个单一的文件 refman.rtf 中。此文件为送入微软 word 以作了优化。某些信息采用了域编码，希望获得实际的数值，可选择所有并触发域（下拉菜单中的选项，并点击鼠标右键即可）。

XML 输出

它包含一个聚合信息的“堆”结构，每一个单元 compound（类似 class/namespace/file）都有它自己的 XML 文件以及 index.xml 的索引文件。

也会生成一个 .xslt 的 XSLT 脚本，用于组合所有的 XML 文件到一个单一的文件中。

能生成两个 XML 图式文件，index.xsd（用于 index 文件），compound.xsd（用于单元 compound 文件）。图式文件用于描述存在的元素，元素的属性以及如何结构它们，它使用的是 XML 文件的语法，可用于验证或者控制 XSLT 脚本。

在 addon/doxmlparser 目录中你能找到一个解析器的库，它可读取对 XML 文件以增量方式进行处理的输出（查阅 addon/doxmlparser/include/doxmlintf.h 中包含有库接口的说明）。

Man page 输出

使用 man 程序查看生成 man 页面，你必须确认 man 目录是否被放置到 man 路径下（查看 MANPATH 环境变量），注意一下 man 格式中的一些功能限制，有部分信心可能会丢失（象类图表，交叉引用，公式）。

第三步：源码文档化

尽管第三步才出现源码的文档化操作，但是对一个新项目而言现在只是前两步的延续。这里假定你已经有了一份代码，并且希望通过 doxygen 来生成一份漂亮的文档，来描述 API 以及代码内部。

如果 EXTRACT_ALL 设置为 NO（默认值），doxygen 只生成一个包含文档化后的成员，源文件，类以及名字空间的

记录。以下是成员，类以及名字空间在文档化中基本的两种组合方式，你会如何选择呢？

1. 放置一个特殊文档块（Special documentation blocks），在成员，类以及名字空间的声明或定义的前面。允许文档记录直接放置在成员之后 For file, class and namespace members it is also allowed to place the documention directly after the member，查阅“**特殊文档块**”节，来获得特殊文档块更多的信息。

2. 放置一个特殊文档块在任意处（可以是另一个文件或其余位置），并在文档块中放置一个结构化命令字（structural command），结构化命令字指向文档块中能被文档化的一些部分（比如一个成员，类，名字空间或者源文件），查阅“**放置在其他位置的文档**”节，学习更多结构命令字的内容。

使用上面第二个选项，只有源文件能被文档化，因为文件前端无法放置一个文档块，那么源文件成员（函数，变量，重命名，定义）不需要一个详细的结构化命令，只要放置一个特殊文档块在他们前端或后端。

一个特殊文档块在它们写入 HTML 或 LaTex 输出文件之前将被解析。

解析过程的步骤如下：

1. 文档化过程中特殊命令将被执行，查阅“**特殊命令字**”章节中所有命令的概览。
2. 如果一行是由若干空白和星号开始，那么这些空白和星号都将被删除。
3. 所有空白行都将被视为段落分隔符，它能保证放入新的分段命令，生成可读出的文档。
4. 创建相同名称的链接，为文档化一个类做准备（除非名称前有一个“%” ，那么此名称无法建立连接，且“%”号将被删除 ）。
5. 如在文本找到某些规则，成员的连接将被创建，查阅“**自动连接生成**”获得如何创建自动连接的更多信息。
6. 文档中的 HTML 标记能被解释和转换成 LaTex 而输出，查阅“HTML 命令字”获得所有 HTML 标记的概览。

代 码 文 档 化

特殊文档块

一个特殊文档块其实是一个带有若干标记的 C/C++风格的注释块，doxygen 清楚特殊文档块是在生成文档中需要被出除的那一部分。对于 Python 和 VHDL 代码中不同的注释约定，可以在“**Python 的特殊文档块**”和“**VHDL 的特殊文档块**”的章节找到各自的信息。

对于任何一行代码来说，会有两种（也可能是三种）注释方式，并且会一同放置到文档中，一种是简明描述，另一种是细节描述，这两种方式都是可选的。而方法和函数也可能使用第三种描述的方式，它被称为\“in body\”描述，并包含方法和函数内所有注释块之间的联系。

多于一个的简明或者细节描述是能被接受的（不推荐，将会无法判定描述的顺序）。

建议简明描述使用一个短小的单行方式，而细节描述要提供足够的长度以包容更多的细节文档。一个“in body 描述可看成一个细节描述，或者是执行细节的集合。以 HTML 方式输出的简明描述可在文档行引用之处支持冒泡提示 For the HTML output brief descriptions are also use to provide tooltips at places where an item is referenced 。

这里有若干方法，用于标记一个注释块为细节描述：

1. 可使用 JavaDoc 风格，包含两个“*”开头的 C 注释风格，如同：

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

2. 可使用 QT 风格，即在 C 注释风格基础上添加一个！字符，例子如下：

```
/*!
* ... text ...
*/
```

中间的*号是可选的，下面的例子是有效的。

```
/*!
... text ...
*/
```

3. 第三种替代方式则使用至少两行的C++注释块，在每行开始出添加一个斜杠或者惊叹号，这有两个例子：

```
///
/// ... text ...
///
```

```
//!
//!... text ...
//!
```

这种情况下，请使用一条空行作为文档块的结尾。

4. 有些人喜欢在文档中使用他们自己的注释块，感觉更清晰。基于这样的目的，可能会用到以下的方式：

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

（注意前两条斜杠结束普通注释块，后两条开始一个特殊注释块）

```
//////////////////////////////////////////////////
/// ... text ...
//////////////////////////////////////////////////
```

简明描述可能出现的若干情况：

1.在一个注释块之前使用\brief 命令。这个命令能在一个段落的末尾处终止它，并在一条空白行之后紧跟细节描述。这有一个例子。

```
/*! \brief Brief description.
*         Brief description continued.
*
*  Detailed description starts here.
*/
```

2.如果在配置文件中将 JAVADOC_AUTOBRIEF 设置为 YES，可使用 JavaDoc 注释块，并自动展开一个简明描述，在出现第一个“.” 处结束，后跟一个空格或空白行。这里有一个例子：

```
/** Brief description which ends at this dot. Details follow
*  here.
*/
```

这个选项标记对多行的C++注释块同样有效：

```
/// Brief description which ends at this dot. Details follow
```

```
/// here.
```

3.用于不超过一行的C++注释块 A third option is to use a special C++ style comment which does not span more than one line，这有两个例子。

```
/// Brief description.
/** Detailed description. */
```

或者

```
//! Brief descripion.

//! Detailed description
//! starts here.
```

注意上一个例子中空行，用于分隔注释块中简明描述和细节描述，此类情况下将 JAVADOC_AUTOBRIEF 设置为 NO。

你会发现 doxygen 相当灵活，如果有多行细节描述，可参考以下的例子：

```
//! Brief description, which is
//! really a detailed description since it spans multiple lines.
/*! Another detailed description!
*/
```

多个细节描述被组合在一起，请注意假如这些细节描述被放置到代码的不同地方，那么能否找到只能取决于 doxygen 对代码的解析程度而定。Note that this is also the case if the descriptions are at different places in the code! In this case the order will depend on the order in which doxygen parses the code.

这有一个 C++代码块使用 QT 注释风格的例子：

```
//!  A test class.
/*!
  A more elaborate class description.
*/

class Test
{
  public:

    //! An enum.
    /*! More detailed enum description. */
    enum TEnum {
                 TVal1, /*!< Enum value TVal1. */
```

```
                TVal2, /*!< Enum value TVal2. */
                TVal3  /*!< Enum value TVal3. */
              }
        //! Enum pointer.
        /*! Details. */
        *enumPtr,
        //! Enum variable.
        /*! Details. */
        enumVar;

    //! A constructor.
    /*!
      A more elaborate description of the constructor.
    */
    Test();

    //! A destructor.
    /*!
      A more elaborate description of the destructor.
    */
   ~Test();

    //! A normal member taking two arguments and returning an integer value.
    /*!
      \param a an integer argument.
      \param s a constant character pointer.
      \return The test results
      \sa Test(), ~Test(), testMeToo() and publicVar()
    */
    int testMe(int a,const char *s);

    //! A pure virtual member.
    /*!
      \sa testMe()
      \param c1 the first argument.
      \param c2 the second argument.
    */
```

```
    virtual void testMeToo(char c1,char c2) = 0;

    //! A public variable.
    /*!
      Details.
    */
    int publicVar;

    //! A function variable.
    /*!
      Details.
    */
    int (*handler)(int a,int b);
};
```

http://www.stack.nl/~dimitri/doxygen/docblocks.html

通常一行注释只会有一个简明描述，而多行注释块会包含一个更详细的描述。

类，名字空间或文件的成员概述中会包含简明描述，可使用 small italic 字体打印（这类描述能被隐藏，设置 BRIEF_MEMBER_DESC 为 NO）。默认情况下简明描述作为细节描述的第一条句子（可通过设定 REPEAT_BRIEF 为 NO 改变），在 QT 风格中简明描述和细节描述都是可选的。

默认状态，JavaDoc 文档块与 QT 文档块规则类似，JavaDoc 不符的地方是，它会将文档块的第一句自动当成是简明描述，可通过设置 JAVADOC_AUTOBRIEF 为 YES 使能此动作。如果你有效了此选项，需要在句子中放置一个“.”（点）来结束句子，并在点之后放置一个反斜杠或空格，这有一个例子：

```
/** Brief description (e.g.\ using only a few words). Details follow. */
```

这里有一个与前面类似的代码片段，使用 JavaDoc 风格且 JAVADOC_AUTOBRIEF 为 YES：

```
/**
 *  A test class. A more elaborate class description.
 */

class Test
{
  public:

    /**
     * An enum.
```

```
     * More detailed enum description.
     */

    enum TEnum {
           TVal1, /**< enum value TVal1. */
           TVal2, /**< enum value TVal2. */
           TVal3  /**< enum value TVal3. */
         }
       *enumPtr, /**< enum pointer. Details. */
       enumVar;  /**< enum variable. Details. */

      /**
       * A constructor.
       * A more elaborate description of the constructor.
       */
      Test();

      /**
       * A destructor.
       * A more elaborate description of the destructor.
       */
     ~Test();

      /**
       * a normal member taking two arguments and returning an integer value.
       * @param a an integer argument.
       * @param s a constant character pointer.
       * @see Test()
       * @see ~Test()
       * @see testMeToo()
       * @see publicVar()
       * @return The test results
       */
       int testMe(int a,const char *s);

      /**
       * A pure virtual member.
```

```
        * @see testMe()
        * @param c1 the first argument.
        * @param c2 the second argument.
        */
        virtual void testMeToo(char c1,char c2) = 0;

       /**
        * a public variable.
        * Details.
        */
        int publicVar;

       /**
        * a function variable.
        * Details.
        */
        int (*handler)(int a,int b);
};
```

http://www.stack.nl/~dimitri/doxygen/examples/jdstyle/html/class_test.html

同样如果希望 QT 文档块自动将第一条句子作为简明描述，设置 QT_AUTOBRIEF 为 YES。

与其他大多数文档系统不同，doxygen 允许你在定义的前端放置文档成员（包括全局函数）。这种方法使得文档能放在源文件而不是头文件中，它保持了头文件的兼容性，允许成员的执行者直接访问文档。折中的方法是简明描述放在声明之前，而细节描述放在成员定义之前。

在成员之后放置文档

如果你希望文档化一个文件，结构，联合，类以及枚举的成员，需要为复合体中的这些成员放置文档，有时希望放置在成员之后的文档块，可以替换掉原有的块，因此要在注释块中加入“<” 标记。请注意“<”作为一个函数参数值是有效的 Note that this also works for the parameters of a function 。

这有一些例子：

```
int var; /*!< Detailed description after the member */
```

在一个成员之后放置一个 QT 风格的细节文档块，其他方式也可以做到这一点：

```
int var; /**< Detailed description after the member */
```

或

```
int var; //!< Detailed description after the member
         //!<
```

或

```
int var; ///< Detailed description after the member
         ///<
```

大多数时候也需要在一个成员之后，放置一个简明描述，如下：

```
int var; //!< Brief description after the member
```

或

```
int var; ///< Brief description after the member
```

函数中可用“@param”来文档化它的参数，使用“[in], [out], [in,out]”文档化它的执行方向。内联文档则可能从方向属性开始记录 For inline documentation this is also possible by starting with the direction attribute，例如：

```
void foo(int v /**< [in] docs for input parameter v. */);
```

要注意的是，这些块都有相同的结构和意义，如同前一章中所介绍的特殊注释块，只使用“<”指示，成员将前一个注释块替换后一个注释块。这有一个使用注释块的例子：

```
/*! A test class */

class Test
{
  public:
    /** An enum type.
     *  The documentation block cannot be put after the enum!
     */
    enum EnumType
    {
      int EVal1,     /**< enum value 1 */
      int EVal2      /**< enum value 2 */
    };
    void member();   //!< a member function.

  protected:
```

```
    int value;       /*!< an integer value */
};
```

http://www.stack.nl/~dimitri/doxygen/examples/afterdoc/html/class_test.html

警告：

这些块只能用于文档化成员和参数，无法用于文件，类，联合，结构，组，名字空间以及枚举，此外下一节提到的结构化命令（比如\class）是不允许在这些注释块中出现的。

其余位置的文档

到目前为止，我们假定文档块总是位于文件，类，名字空间的声明或定义的前端，或者处于它们成员的前后注释块中。尽管大多数时间这个约束是合适的，但有时出于别的原因要将文档放到其他位置。因为文件中不存在“文件前端”的位置而需要对文件进行文档化 For documenting a file this is even required since there is no such thing as "in front of a file" 。

Doxygen 允许你将文档块放到任何位置（一个函数或一个 C 注释块的主体，是一个例外）。

当文本行的文档块中需要一个结构化命令，那么你所付出的代价是，在文本行的前后端不能直接放置文档块，因为它有可能携带一些信息的副本。所以在实际中你最好避免使用结构化命令，除非其他的需求迫使你这样做。

结构化命令（和其他所有的命令一样）由一个反斜杠或“@”开始，如果你喜欢 JavaDoc 风格，紧跟一个命令字和若干参数。比如你想文档化例子（http://www.stack.nl/~dimitri/doxygen/examples/）中的 Test 类，你要在 doxygen 读取的输入端的某个位置放入文档块 you could have also put the following documentation block somewhere in the input that is read by doxygen ：

```
/*! \class Test
\brief A test class.


A more detailed class description.
*/
```

这有一个特殊命令 \class 用于指示注释块包括类 Test 的文档。其他的结构化命令字：

\struct 文档化一个 C 结构
\union 文档化一个联合
\enum 文档化一个枚举
\fn 文档化一个函数
\var 文档化一个变量，类型转换，枚举值其中之一
\def 文档化一个#define
\typedef 文档化一个类型转换

\file 文档化一个文件
\namespace 文档化一个名字空间
\package 文档化一个 Java 包
\interface 文档化一个 IDL 接口

查阅“**特殊命令字**”节获取更多命令字信息。

文档化一个 C++类成员，你必须先文档化 C++类，这同样适用于名字空间。文档化一个全局 C 函数，类型转换，枚举以及预处理定义，你必须先文档化包含以上内容的文件（通常它是一个头文件，因此才能将其包含的信息输送给其他源文件）。

让我们重复一次，因为它经常被忽略：文档化全局对象（函数，类型转换，枚举，宏，等等），你必须先文档化包含它们的文件，换句话说，必须在文件至少有一行以下内容：

```
/*! \file */
```

或

```
/** @file */
```

这有一个使用结构化命令进行文档化的 C 头文件 structcmd.h：

```
/*! \file structcmd.h
    \brief A Documented file.

    Details.
*/

/*! \def MAX(a,b)
    \brief A macro that returns the maximum of \a a and \a b.

    Details.
*/

/*! \var typedef unsigned int UINT32
    \brief A type definition for a .

    Details.
*/

/*! \var int errno
    \brief Contains the last error code.
```

```
    \warning Not thread safe!
*/

/*! \fn int open(const char *pathname,int flags)
    \brief Opens a file descriptor.

    \param pathname The name of the descriptor.
    \param flags Opening flags.
*/

/*! \fn int close(int fd)
    \brief Closes the file descriptor \a fd.
    \param fd The descriptor to close.
*/

/*! \fn size_t write(int fd,const char *buf, size_t count)
    \brief Writes \a count bytes from \a buf to the filedescriptor \a fd.
    \param fd The descriptor to write to.
    \param buf The data buffer to write.
    \param count The number of bytes to write.
*/

/*! \fn int read(int fd,char *buf,size_t count)
    \brief Read bytes from a file descriptor.
    \param fd The descriptor to read from.
    \param buf The buffer to read into.
    \param count The number of bytes to read.
*/

#define MAX(a,b) (((a)>(b))?(a):(b))
typedef unsigned int UINT32;
int errno;
int open(const char *,int);
int close(int);
size_t write(int,const char *, size_t);
int read(int,char *,size_t);
```

http://www.stack.nl/~dimitri/doxygen/examples/structcmd/html/structcmd_8h.html

因为例子中的每一个注释块中都包含有一个结构化命令字，在不影响生成文档的情况下，所有的注释块将被移动到另一个位置或者是输入文件（例如源文件）。这种方式不利的一面，所有的原型都要复制，那么所有的变更都需要执行两次！正因为如此你首先考虑上述的方式是否真的需要，和避免可能出现结构化命令。我经常收到一些例子，注释块中包含\fn 命令并放置在函数前端。很明显\fn 命令字是多余的，而它只会带来问题。

Python 中的特殊文档块

Python 文档化代码的标准方式，调用文档字串来完成。字串可以保存在__doc__中，并能实时检索。Doxygen 能引用这些注释，并假定他们在预定义格式中可以被描述 Doxygen will extract such comments and assume they have to be represented in a preformatted way 。

```
"""@package docstring
Documentation for this module.

More details.
"""

def func():
    """Documentation for a function.

    More details.
    """
    pass

class PyClass:
    """Documentation for a class.

    More details.
    """

    def __init__(self):
        """The constructor."""
        self._memVar = 0;
```

```
    def PyMethod(self):
        """Documentation for a method."""
        pass
```

http://www.stack.nl/~dimitri/doxygen/examples/docstring/html/index.html

注意，在此模式下 doxygen 的特殊命令字是不支持的。

另一种文档化 Python 代码的方法使用“##”开始，这种注释块类型与其他 doxygen 支持语言的文档块比较一致，在这种模式下允许使用特殊命令字 These type of comment blocks are more in line with the way documentation blocks work for the other languages supported by doxygen and this also allows the use of special commands。

这是使用 doxygen 风格注释的同一个例子：

```
## @package pyexample
#  Documentation for this module.
#
#  More details.

## Documentation for a function.
#
#  More details.
def func():
    pass

## Documentation for a class.
#
#  More details.
class PyClass:

    ## The constructor.
    def __init__(self):
        self._memVar = 0;

    ## Documentation for a method.
    #  @param self The object pointer.
    def PyMethod(self):
        pass
```

```
    ## A class variable.
    classVar = 0;

    ## @var _memVar
    #  a member variable
```

http://www.stack.nl/~dimitri/doxygen/examples/pyexample/html/index.html

因为 Python 看上去越来越象 Java 而不是 C/C++，所以需要设置 OPTMIZE_OUTPUT_JAVA 为 YES。

VHDL 中的特殊文档块

VHDL 的注释通常从“--”开始，doxygen 的注释从“--!”开始。在 VHDL 中只有这两种注释类型，“--!”开头的注释表明它是一个简明描述，而一个多行“--!”开头（每一行开始都会有“--!”符号）的注释则表明它是一个细节描述。

注释总是处于将被文档化的文本行前端，并存在一个例外：端口的注释处于文本行的后端，并被当作端口的一个简明描述。

这有一个使用 doxygen 注释的 VHDL 文件的例子：

```
------------------------------------------------
--! @file
--! @brief 2:1 Mux using with-select
------------------------------------------------

--! Use standard library
library ieee;
--! Use logic elements
    use ieee.std_logic_1164.all;

--! Mux entity brief description

--! Detailed description of this
--! mux design element.
entity mux_using_with is
```

```
    port (
        din_0   : in  std_logic; --! Mux first input
        din_1   : in  std_logic; --! Mux Second input
        sel     : in  std_logic; --! Select input
        mux_out : out std_logic  --! Mux output
    );
end entity;

--! @brief Architure definition of the MUX
--! @details More details about this mux element.
architecture behavior of mux_using_with is
begin
    with (sel) select
    mux_out <= din_0 when '0',
               din_1 when others;
end architecture;
```

http://www.stack.nl/~dimitri/doxygen/examples/mux/html/index.html

为获得合适的输出，需要设定 OPTIMIZE_OUTPUT_VHDL 为 YES，这个设置将会影响其他的一些设置。当他们没有被正确设定时，doxygen 也能发出一个警示，告知在哪里如何更改设置。

列　　表

Doxygen 支持创建文本行列表的多种方法。

使用破折号

在每一行的开始处放置列对齐的多个减号“-”，一个 bullet 列表将自动生成。使用“-”紧跟“#”，也能生成一个带编号的列表。

这有一个例子：

```
/*!
   *  A list of events:
   *    - mouse events
   *         -# mouse move event
   *         -# mouse click event\n
   *            More info about the click event.
   *         -# mouse double click event
   *    - keyboard events
   *         -# key down event
   *         -# key up event
   *
   *  More text here.
   */
```

最终结果是：

A list of events:

- mouse events
  a. mouse move event
  b. mouse click event
     More info about the click event.

```
        c. mouse double click event
    ● keyboard events
        a. key down event
        b. key up event
More text here.
```

如果在列表中使用 tabs 进行缩排，请确认配置文件中 TAB_SIZE 选项是否设置了正确的 tab 尺寸。
可在列表结束的缩排层级的空白处放置一个点“.”或者开始一个新的段落，即可结束一个列表。
这有一个不言自明的例子：

```
/**
 * Text before the list
 * - list item 1
 *   - sub item 1
 *     - sub sub item 1
 *     - sub sub item 2
 *     .
 *     The dot above ends the sub sub item list.
 *     More text for the first sub item
 *   .
 *   The dot above ends the first sub item.
 *   More text for the first list item
 *   - sub item 2
 *   - sub item 3
 * - list item 2
 * .
 * More text in the same paragraph.
 *
 * More text in a new paragraph.
 */
```

使用 HTML 命令

如果你喜欢也可以在文档块中使用 HTML 命令。使用这些命令的优点在于，列表行中包含的多个段落看起来更自然。

这有一个运用 HTML 命令的例子：

```
/*!
   *  A list of events:
   *  <ul>
   *  <li> mouse events
   *     <ol>
   *     <li>mouse move event
   *     <li>mouse click event\n
   *         More info about the click event.
   *     <li>mouse double click event
   *     </ol>
   *  <li> keyboard events
   *     <ol>
   *     <li>key down event
   *     <li>key up event
   *     </ol>
   *  </ul>
   *  More text here.
   */
```

注意：

在 HTML 模式下缩排的层级并不重要。

使用 \arg 或者 @li

为兼容 QT 的内部文档工具 qdoc 和 kDoc，doxygen 有两个可用于创建简单的无嵌套列表的命令。
查看以下网页获得更多信息。

\arg ： http://www.stack.nl/~dimitri/doxygen/commands.html#cmdarg
\li：http://www.stack.nl/~dimitri/doxygen/commands.html#cmdli

组　　合

Doxygen 一共有三种组合的机制，一种工作于全局级别，为每一个组创建一个新页面。在文档中这些组被称为“**模块**”，第二种工作在一些复合体的成员列表中，参考“**成员组**”。第三种工作于页面，参考“**子页面**”。

模块

模块是在一个独立页面中进行组合的方法，你可以文档化整个组以及所有单独的成员，一个组的成员可以是文件，名字空间，类，函数，变量，枚举，类型转换以及定义，但也可以是其他的组。

定义一个组，你需要在特殊注释块中放置\defgroup 命令。命令的第一个参数是一个组的 uid（uniquely identify）label，第二个参数是将出现在文档中的组的名称或标题。

你可以建立一个组中的成员个体，通过在组的文档块中放置\ingroup 命令。

在组之前使用打开标记“@{”，或在组之后使用关闭标记“@}”，将每一个成员组合在一起，并在文档中避免放置\ingroup 命令。这些标记可放置在组定义的文档或一个独立的文档块中。

组可以使用自身的组标记实现嵌套。

当使用相同的组 label 超过一次，你将得到一个错误消息。如果你不希望 doxygen 强制组 label 的唯一性，可用\addtogroup 替换掉\defgroup，当组已经被定义，它可以合并到一个已经存在的文档中，形成一个新组。组标题对于这个命令来说是可选的。所以你可以使用

```
/** \addtogroup <label> */
/*\@{*/
/*\@}*/
```

添加成员到一个已定义组的更多细节另有描述。

注意：复合体（比如类，文件，名字空间）可放到多个组中，而成员（比如变量，函数，类型转换，枚举）只能是一个组中的组成员（这种位置上的限制，是为了当一个成员在它的类，名字空间，文件的正文中无法被文档化而在组中却是可见的情况下，避免链接组目标时产生二义性 this restriction is in place to avoid ambiguous linking targets in case a member is not documented in the context of its class, namespace or file,

but only visible as part of a group)。

Doxygen 放置组成员时，会将其定义为最高的“优先级”：例如一个直接\ingroup 命令可通过“@{ @}”覆盖掉一个意义含混的组定义。相同优先级下的组定义冲突将触发一个警告，除非一个成员定义并未出现在其他直接的文档中。

以下的例子是放置 VarInA 到组中，解决 IntegerVariable 放置到 IntVariables 组的冲突，因为 extern int IntegerVariable 无法不被文档化的：The following example puts VarInA into group A and silently resolves the conflict for IntegerVariable by putting it into group IntVariables, because the second instance of IntegerVariable is undocumented

```
/**
 * \ingroup A
 */
extern int VarInA;

/**
 * \defgroup IntVariables Global integer variables
 */
/*@{*/

/** an integer variable */
extern int IntegerVariable;

/*@}*/

....

/**
 * \defgroup Variables Global variables
 */
/*@{*/

/** a variable in group A */
int VarInA;

int IntegerVariable;

/*@}*/
```

\ref 命令定义组的引用名称，第一个参数\ref 命令将用到组 label，如果要用自定义链接名称，可在 label 之后可放置使用双引号修饰的链接名称，如下面的例子所示：

```
This is the \ref group_label "link" to this group.
```

组定义的优先级（从最高到最低）：\ingroup,\defgroup,\addtogroup,\weakgroup。最后的命令与\addtogroup 想比有一个更低的优先级（？），但\weakgroup 允许你添加“松散 lazy”的组定义：你能在.h 文件中使用更高级别的命令来定义层次结构，而在.c 文件中的\weakgroup 无须完全复制这种层次结构。

例子：

```
/** @defgroup group1 The First Group
 *  This is the first group
 *  @{
 */

/** @brief class C1 in group 1 */
class C1 {};

/** @brief class C2 in group 1 */
class C2 {};

/** function in group 1 */
void func() {}

/** @} */ // end of group1

/**
 *  @defgroup group2 The Second Group
 *  This is the second group
 */

/** @defgroup group3 The Third Group
 *  This is the third group
 */

/** @defgroup group4 The Fourth Group
 *  @ingroup group3
 *  Group 4 is a subgroup of group 3
```

```
 */
```

```
/**
 *  @ingroup group2
 *  @brief class C3 in group 2
 */
class C3 {};

/** @ingroup group2
 *  @brief class C4 in group 2
 */
class C4 {};

/** @ingroup group3
 *  @brief class C5 in @link group3 the third group@endlink.
 */
class C5 {};

/** @ingroup group1 group2 group3 group4
 *  namespace N1 is in four groups
 *  @sa @link group1 The first group@endlink, group2, group3, group4
 *
 *  Also see @ref mypage2
 */
namespace N1 {};

/** @file
 *  @ingroup group3
 *  @brief this file in group 3
 */

/** @defgroup group5 The Fifth Group
 *  This is the fifth group
 *  @{
 */

/** @page mypage1 This is a section in group 5
```

```
 *  Text of the first section
 */

/** @page mypage2 This is another section in group 5
 *  Text of the second section
 */

/** @} */ // end of group5

/** @addtogroup group1
 *
 *  More documentation for the first group.
 *  @{
 */

/** another function in group 1 */
void func2() {}

/** yet another function in group 1 */
void func3() {}

/** @} */ // end of group1
```

模块参考：
http://www.stack.nl/~dimitri/doxygen/examples/group/html/modules.html

成员组

如果一个复合体（比如一个类或文件）有很多成员，通常希望能将成员们一同合并到组中，doxygen 在类型和保护层级之上，已经能够自动组合， 但是你也许感觉有这些还不够或是认为缺省的组合操作将会出错，例如你觉得存在类型差异（语法的）的成员属于相同（语义上的）的组。

如果你喜欢 C 风格注释，用以下两种块模板来定义一个成员组，注意成员组内部的组成员都必须是确实存在。

```
//@{
  ...
```

```
//@}
```

或

```
/*@{*/
...
/*@}*/
```

在一个块开始标记“@{”之前可以放置一个单独的注释块，而此块必须包含@name (或 \name)命令，用于指定组的头部，也可选择让注释块包含更多的组信息。

成员组的嵌套是不被允许的。

如果一个类的成员组中的全部组成员都有相同的类型和保护级别（比如全是静态公有成员），那么整个成员组被当成类型/保护级别组的一个子分组（例如成员组将被看\“静态公有成员\”节点的子节点），假如两个或多个成员有不同的类型，那么组会将它们置于相同的层级上自动生成一个组，假如你希望强制类的全部组成员在顶层，你需要在类文档中加入\nosubgrouping 命令。

例子：

```
/** A class. Details */
class Test
{
  public:
    //@{
    /** Same documentation for both members. Details */
    void func1InGroup1();
    void func2InGroup1();
    //@}

    /** Function without group. Details. */
    void ungroupedFunction();
    void func1InGroup2();
  protected:
    void func2InGroup2();
};

void Test::func1InGroup1() {}
void Test::func2InGroup1() {}
/** @name Group2
 *  Description of group 2.
```

```
 */
//@{
/** Function 2 in group 2. Details. */
void Test::func2InGroup2() {}
/** Function 1 in group 2. Details. */
void Test::func1InGroup2() {}
//@}

/*! \file
 *  docs for this file
 */

//@{
//! one description for all members of this group
//! (because DISTRIBUTE_GROUP_DOC is YES in the config file)
#define A 1
#define B 2
void glob_func();
//@}
```

http://www.stack.nl/~dimitri/doxygen/examples/memgrp/html/class_test.html

Group1 看成公有成员的子节点，而 Group2 则是一个独立节点，因为它包含成员中存在不同保护级别（即公有和保护）。

子页面

信息可用\page 和\mainpage 命令合并到页面中，正常情况下，结果将置于页面的平直（无缩排）列表中，列表的起始端即为\“主\”页面。

在“**模块**”章节中简述添加结构的替代方法，为了更自然和方便地增加额外结构到页面中，可以使用\subpage 命令。

A 页面用\subpage 命令增加一个链接到 B 页面，同时将 B 页面变成 A 页面的一个子页。这对于 GA，GB 两个组同样有影响，使得 GB 变为 GA 一部分的方法，将 A 页面置于 GA 组内，而将 B 页面置于 GB 组内即可。

包 含 公 式

Doxygen 允许你放置 Latex 公式在输出中(只能支持 HTML,Latex 的输出,无法用于 RTF 和 man 页面输出),在 HTML 文档中可以包含公式(可看成图片),你需要安装以下工具:

Latex: latex 编译器,解析公式时需要它,我使用 teTex 1.0 发行版本做测试
dvips: 转换 DVI 文件为 PostScript 文件的工具,我使用 Radical Eye 5.92 版本做测试
gs: Ghostscript 解释器,用于转换 PostScript 文件为位图,我使用 Aladdin Ghostscript 8.0 做测试

在文档中包含公式的若干方法:

1. 在运行的文本中使用文本公式,这些公式可以放置在一对 \f$ 命令之间,如:

```
The distance between \f$(x_1,y_1)\f$ and \f$(x_2,y_2)\f$ is
\f$\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}\f$.
```

结果为:

(x_1,y_1) 与 (x_2,y_2) 之间的距离为 $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

2. 集中于单独一行的非数字公式,这些公式可放置在 \f[和 \f]命令之间,一个例子:

```
\f[
    |I_2|=\left| \int_{0}^T \psi(t)
            \left\{
               u(a,t)-
               \int_{\gamma(t)}^a
               \frac{d\theta}{k(\theta,t)}
               \int_{a}^\theta c(\xi)u_t(\xi,t)\,d\xi
            \right\} dt
         \right|
\f]
```

结果为:

$$|I_2|=\left|\int_0^T \psi(t)\left\{u(a,t)-\int_{\gamma(t)}^a \frac{d\theta}{k(\theta,t)}\int_a^\theta c(\xi)u_t(\xi,t)\,d\xi\right\}dt\right|$$

3. 对于非数学领域的公式或 Latex 元件可用\f{environment}命令说明, environment 是 Latex 所支持的 environment 名称,配合\f}终止命令,这有一个等式计算的例子:

```
\f{eqnarray*}{
        g &=& \frac{Gm_2}{r^2} \\
          &=& \frac{(6.673 \times 10^{-11}\,\mbox{m}^3\,\mbox{kg}^{-1}\,
              \mbox{s}^{-2})(5.9736 \times
10^{24}\,\mbox{kg})}{(6371.01\,\mbox{km})^2} \\
          &=& 9.82066032\,\mbox{m/s}^2
\f}
```

结果为：

$$
\begin{eqnarray*}
g &=& \frac{Gm_2}{r^2} \\
&=& \frac{(6.673 \times 10^{-11}\,\mathrm{m}^3\,\mathrm{kg}^{-1}\,\mathrm{s}^{-2})(5.9736 \times 10^{24}\,\mathrm{kg})}{(6371.01\,\mathrm{km})^2} \\
&=& 9.82066032\,\mathrm{m/s}^2
\end{eqnarray*}
$$

前两个命令都能确定公式包含有LaTeX数学模式下的有效命令，第三个命令包含特殊环境下的有效命令。

警告：

当前 doxygen 的公式从错误中自动修正的容错能力，还不能令人满意。它可能会成功删除已写入 html 目录的 formula.repository 文件，以丢弃一个出错的公式。

图 形 与 图 表

Doxygen 已经内建了对 C++类提供继承关系图表创建的支持。

Doxygen 可以使用 graphviz 中的点工具生成更高级图表和图形，Graphviz 是一个开源，跨平台的图形绘制工具箱，能从 http://www.graphviz.org/ 找到它。

假若在系统路径你可以找到点工具，在配置文件中设置 HAVE_DOT 为 YES 让 doxygen 使用它。

Doxygen 使用点工具创建以下图形：

1. 在原文中绘制一个图形化表示的类层次，当前这个特性只支持 HTML。
   警告：
   当你有一个非常巨大的类层次，并且许多类来自于一个公共的基类，那么得到图片可能变得非常大，需要为某些浏览器进行图片处理。

2. 继承图在每一个类文档都会生成，它能显示直接和间接的继承关系，它能禁用内建的继承图表的生成。
3.
4. 依赖包容图在每一个文件的文档中都会生成，且文件中至少要包含一个以上其他文件，这个特性当前只支持 HTML 和 RTF。

5. 每一个类和结构的图表中显示：
   基类的继承关系
   与其他类和结果的使用关系（比如，类 A 有一个成员变量 m_a，m_a 作为类 B 的一个类型，那么通过类似于标签的 m_a，A 将有一个箭头指向 B）

6. 如果 CALL_GRAPH 设置为 YES，将为每一个函数绘制一个调用图形，它能够显示函数直接和间接的调用。

7. 如果 CALLER_GRAPH 设置为 YES，将为每一个函数绘制一个调用图形，它能够显示函数直接和间接的被谁所调用。

使用一个布局文件你可以确认实际显示的是那一个图形，参看“**自定义输出**”章节。

选项 DOT_GRAPH_MAX_NODES，MAX_DOT_GRAPH_DEPTH 用于限制生成图形的尺寸。

以下是 HTML，RTF 中类图表的元素意义：

○ 黄色框表示一个类，框内右下角有一个小标记，显示类所含的被隐藏的基类。当前类图表最大的树节点为 8 个，如果树比较大那么有些节点将被隐藏。如果是虚线框，那表明是虚继承。

- 白色框表明当前显示的是类文档
- 灰色框表明它是一个未文档化的类
- 实线蓝色箭头表明是公有继承
- 虚线绿色箭头表明是保护继承
- 点线绿色箭头表明是私有继承

以下是LaTeX中类图表的元素意义：

- 白色框表示一个类，框内右下角有一个小标记，显示类所含的被隐藏的基类。如果是虚线框，那表明是虚继承。
- 实线箭头表明是公有继承
- 虚线箭头表明是保护继承
- 点线箭头表明是私有继承

以下是使用点工具生成图形的元素意义：

- 白色框可用表示一个类，结构或者文件
- 红色框表明它是拥有许多箭头的节点，换言之，它是只关心此节点的整个图形的一部分。原因是整个图形是非常大的，有时截取当前需要显示的那一部分。Doxygen 会尝试限制点生成的图形的宽度为 1024 像素。
- 黑色框表明当前显示的是类文档
- 黑线蓝色箭头表明是包容关系（图形的依赖包容）或是公有继承。
- 黑线绿色箭头是保护继承
- 黑线红色箭头是保护继承
- 紫色虚线箭头表明是一个“用法 usage”关系，在箭头的边沿处有变量的关系标签。类 A 使用类 B，假如类 A 有一个成员 m 的类型是 C，那么 B 就是 C 的一个子类型。(比如 C 可能 B，B*，T<B>*)

这有一些头文件，它们能展示出 doxygen 生成多种图表

diagrams_a.h

```
#ifndef _DIAGRAMS_A_H
#define _DIAGRAMS_A_H
class A { public: A *m_self; };
#endif
```

diagrams_b.h

```
#ifndef _DIAGRAMS_B_H
#define _DIAGRAMS_B_H
class A;
class B { public: A *m_a; };
#endif
```

diagrams_c.h

```
#ifndef _DIAGRAMS_C_H
#define _DIAGRAMS_C_H
#include "diagrams_c.h"
class D;
class C : public A { public: D *m_d; };
#endif
```

diagrams_d.h

```
#ifndef _DIAGRAM_D_H
#define _DIAGRAM_D_H
#include "diagrams_a.h"
#include "diagrams_b.h"
class C;
class D : virtual protected  A, private B { public: C m_c; };
#endif
```

diagrams_e.h

```
#ifndef _DIAGRAM_E_H
#define _DIAGRAM_E_H
#include "diagrams_d.h"
class E : public D {};
#endif
```

diagrams_a.h

```
#ifndef _DIAGRAMS_A_H
#define _DIAGRAMS_A_H
class A { public: A *m_self; };
#endif
```

diagrams_b.h

```
#ifndef _DIAGRAMS_B_H
#define _DIAGRAMS_B_H
class A;
```

```
class B { public: A *m_a; };
#endif
```

diagrams_c.h

```
#ifndef _DIAGRAMS_C_H
#define _DIAGRAMS_C_H
#include "diagrams_c.h"
class D;
class C : public A { public: D *m_d; };
#endif
```

diagrams_d.h

```
#ifndef _DIAGRAM_D_H
#define _DIAGRAM_D_H
#include "diagrams_a.h"
#include "diagrams_b.h"
class C;
class D : virtual protected  A, private B { public: C m_c; };
#endif
```

diagrams_e.h

```
#ifndef _DIAGRAM_E_H
#define _DIAGRAM_E_H
#include "diagrams_d.h"
class E : public D {};
#endif
```

diagrams_a.h

```
#ifndef _DIAGRAMS_A_H
#define _DIAGRAMS_A_H
class A { public: A *m_self; };
#endif
```

diagrams_b.h

```
#ifndef _DIAGRAMS_B_H
#define _DIAGRAMS_B_H
class A;
class B { public: A *m_a; };
#endif
```

diagrams_c.h

```
#ifndef _DIAGRAMS_C_H
#define _DIAGRAMS_C_H
#include "diagrams_c.h"
class D;
class C : public A { public: D *m_d; };
#endif
```

diagrams_d.h

```
#ifndef _DIAGRAM_D_H
#define _DIAGRAM_D_H
#include "diagrams_a.h"
#include "diagrams_b.h"
class C;
class D : virtual protected  A, private B { public: C m_c; };
#endif
```

diagrams_e.h

```
#ifndef _DIAGRAM_E_H
#define _DIAGRAM_E_H
#include "diagrams_d.h"
class E : public D {};
#endif
```

http://www.stack.nl/~dimitri/doxygen/examples/diagrams/html/index.html

预　处　理

被使用源文件送入 doxygen 能被内建 C 预处理器解析。

Doxygen 默认情况下只支持局部预处理，它要判定条件编译语句（例如#if），以及宏定义，但它是不能执行宏的展开。

假如你有以下的代码片段

```
#define VERSION 200
#define CONST_STRING const char *

#if VERSION >= 200
  static CONST_STRING version = "2.xx";
#else
  static CONST_STRING version = "1.xx";
#endif
```

默认的 doxygen 将提供以下内容给它的解析器：

```
#define VERSION
#define CONST_STRING

static CONST_STRING version = "2.xx";
```

设置 ENABLE_PREPROCESSING 为 NO 来无效所有预处理，那样 doxygen 将读出两条语句，如：

```
static CONST_STRING version = "2.xx";
static CONST_STRING version = "1.xx";
```

你想要展开 CONST_STRING 宏，设置 MACRO_EXPANSION 为 YES，预处理的结果变为：

```
#define VERSION
#define CONST_STRING

static const char * version = "1.xx";
```

注意：doxygen 会展开所有的宏定义（如果需要递归操作的话）。这种操作通常是太多了，doxygen 也允许只展开那些明确指出的定义，需要设置 EXPAND_ONLY_PREDEF 为 YES，并在 PREDEFINED 或 EXPAND_AS_DEFINED 之后，指定须展开的宏定义。

一个典型的例子，当微软的__declspec 语言扩展行为有效时，一些来自预处理器的帮助则是需要的。A typically example where some help from the preprocessor is needed is when dealing with Microsoft's __declspec language extension 这有一个函数的例子：

```
extern "C" void __declspec(dllexport) ErrorMsg( String aMessage,...);
```

doxygen 将混乱到无事可做，查看__declspec 象是一个函数排序 When nothing is done, doxygen will be confused and see __declspec as some sort of function。以下是 doxygen 典型应用的预处理设置：

```
ENABLE_PREPROCESSING   = YES
MACRO_EXPANSION        = YES
EXPAND_ONLY_PREDEF     = YES
PREDEFINED             = __declspec(x)=
```

确认在 doxygen 解析源代码之前，__declspec(dllexport)已经被删除了。

一个相对复杂的例子，假定你有以下的，一个名为 IUnknown 抽象基类的令人迷惑的代码片段：

```
/*! A reference to an IID */
#ifdef __cplusplus
#define REFIID const IID &
#else
#define REFIID const IID *
#endif


/*! The IUnknown interface */
```

```
DECLARE_INTERFACE(IUnknown)
{
  STDMETHOD(HRESULT,QueryInterface) (THIS_ REFIID iid, void **ppv) PURE;
  STDMETHOD(ULONG,AddRef) (THIS) PURE;
  STDMETHOD(ULONG,Release) (THIS) PURE;
};
```

无论宏是否展开，doxygen 都将陷入混乱，但我们不希望展开 RFIID 宏，因为它是一个文档，当执行接口时，用户需要使用它对文档进行读取。

以下是配置文件的设置：

```
ENABLE_PREPROCESSING = YES
MACRO_EXPANSION      = YES
EXPAND_ONLY_PREDEF   = YES
PREDEFINED           = "DECLARE_INTERFACE(name)=class name" \
                       "STDMETHOD(result,name)=virtual result name" \
                       "PURE= = 0" \
                       THIS_= \
                       THIS= \
                               __cplusplus
```

确认提供给 doxygen 解析器的是正确的结果：

```
/*! A reference to an IID */
#define REFIID

/*! The IUnknown interface */
class  IUnknown
{
  virtual  HRESULT   QueryInterface ( REFIID iid, void **ppv) = 0;
  virtual  ULONG   AddRef () = 0;
  virtual  ULONG   Release () = 0;
};
```

注意 PREDEFINED 标记可接受类似于宏定义的函数（如 DECLARE_INTERFACE），正常的宏替换（如 PURE，THIS），以及简单定义（如__cplusplus）。

注意预处理器的配置由本身自动生成的（如__cplusplus），可由 doxygen 解析器重新定义（因为经常要指定特殊的平台和编译器）。

在一些情况下可能要替换掉一个宏的名字或者函数，由于一些原因无法展开它们的结果给下一步的宏替换。In some cases you may want to substitute a macro name or function by something else without exposing the result to further macro substitution 。要做到这一点，你可使用“：=”操作符而不是“=”。

以下的代码块例子可让我们实现上述的步骤：

```
#define QList QListT
class QListT
{
};
```

使用以下唯一的方法使得 doxygen 如同一个类定义一样解释下面设定，完成 QList 类的定义：Then the only way to get doxygen interpret this as a class definition for class QList is to define:

```
PREDEFINED = QListT:=QList
```

这有一个 Valter Minute 和 Reyes Ponce 提供的例子，它能帮助 doxygen 遍历微软 ATL 与 MFC 库中的模板代码：

```
PREDEFINED             = "DECLARE_INTERFACE(name)=class name" \
                         "STDMETHOD(result,name)=virtual result name" \
                         "PURE= = 0" \
                         THIS_= \
                         THIS= \
                         DECLARE_REGISTRY_RESOURCEID=// \
                         DECLARE_PROTECT_FINAL_CONSTRUCT=// \
                         "DECLARE_AGGREGATABLE(Class)= " \
                         "DECLARE_REGISTRY_RESOURCEID(Id)= " \
                         DECLARE_MESSAGE_MAP= \
                         BEGIN_MESSAGE_MAP=/* \
                         END_MESSAGE_MAP=*/// \
                         BEGIN_COM_MAP=/* \
                         END_COM_MAP=*/// \
                         BEGIN_PROP_MAP=/* \
                         END_PROP_MAP=*/// \
                         BEGIN_MSG_MAP=/* \
                         END_MSG_MAP=*/// \
                         BEGIN_PROPERTY_MAP=/* \
                         END_PROPERTY_MAP=*/// \
                         BEGIN_OBJECT_MAP=/* \
```

```
                        END_OBJECT_MAP()=*/// \
                        DECLARE_VIEW_STATUS=// \
                        "STDMETHOD(a)=HRESULT a" \
                        "ATL_NO_VTABLE= " \
                        "__declspec(a)= " \
                        BEGIN_CONNECTION_POINT_MAP=/* \
                        END_CONNECTION_POINT_MAP=*/// \
                        "DECLARE_DYNAMIC(class)= " \
                        "IMPLEMENT_DYNAMIC(class1, class2)= " \
                        "DECLARE_DYNCREATE(class)= " \
                        "IMPLEMENT_DYNCREATE(class1, class2)= " \
                        "IMPLEMENT_SERIAL(class1, class2, class3)= " \
                        "DECLARE_MESSAGE_MAP()= " \
                        TRY=try \
                        "CATCH_ALL(e)= catch(...)" \
                        END_CATCH_ALL= \
                        "THROW_LAST()= throw"\
                        "RUNTIME_CLASS(class)=class" \
                        "MAKEINTRESOURCE(nId)=nId" \
                        "IMPLEMENT_REGISTER(v, w, x, y, z)= " \
                        "ASSERT(x)=assert(x)" \
                        "ASSERT_VALID(x)=assert(x)" \
                        "TRACE0(x)=printf(x)" \
                        "OS_ERR(A,B)={ #A, B }" \
                        __cplusplus \
                        "DECLARE_OLECREATE(class)= " \
                        "BEGIN_DISPATCH_MAP(class1, class2)= " \
                        "BEGIN_INTERFACE_MAP(class1, class2)= " \
                        "INTERFACE_PART(class, id, name)= " \
                        "END_INTERFACE_MAP()=" \
                        "DISP_FUNCTION(class, name, function, result, id)=" \
                        "END_DISPATCH_MAP()=" \
                        "IMPLEMENT_OLECREATE2(class, name, id1, id2, id3, id4,\
                         id5, id6, id7, id8, id9, id10, id11)="
```

你会发现 doxygen 的预处理器是十分强大的，但也许你需要的更多的灵活性，那在 INPUT_FILTER 标记之后你可以写一个输入过滤器来指定它。

如果你无法确定 doxygen 预处理器的作用，你可以运行以下的命令：

```
doxygen -d Preprocessor
```

它将告知 doxygen 在预处理完成之后，把输入源代码转存到标准输出中。（提示：设定 QUIET = YES, WARNINGS = NO 无效其他的输出）

生 成 自 动 链 接

大多数文档系统都提供了特殊‘参见 see also’功能，可插入文档内其余位置的链接。尽管 doxygen 也有一个命令实现上述功能， 如那一节（查看\sa 节），它允许你放置多种链接在文档的任何地方。LaTeX 文档中的引用是指向一个被写入的页面号来替代一个链接，此外文档末尾的索引可用于快速找到文档中的一个成员，类，名字空间或文件。man 页面中不会生成引用信息。

下一节展示如何生成源文件中多种文档元素的链接。

链接 web 页面和 mail 地址

Doxygen 自动替换在文档中找到的链接（HTML），手动指定链接文本，使用 HTML ‘a’ 标识。

```
<a href="linkURL">link text</a>
```

并能自动转换成其他的输出格式。

链接到类

一个文档化类和文档中与其关联的包含至少一个非小写字符的单词之间，将自动建立一个连接到包含该类页面的链接。放置一个“%”在单词前，即可与相关联的类建立链接。要链接所有的小写字符，使用\ref。

链接到文件

所有词中包含一个点（.）且点不是单词的最终字符，那么该词将被看作是一个文件名，假如该单词与文档化的输入文件同名，那么自动创建一个链接到输入文件。

链接到函数

建立一个到函数的链接，有可能遇到以下几种情况：

1. <functionName>"("<argument-list>")"
2. <functionName>"()"
3. "::"<functionName>
4. (<className>"::")"<functionName>"("<argument-list>")"
5. (<className>"::")"<functionName>"("<argument-list>")"<modifiers>
6. (<className>"::")"<functionName>"()"
7. (<className>"::")"<functionName>

注意：
1. 函数参数必须指定一个正确类型，例如'fun(const std::string&,bool)' 或 'fun()'可匹配一些原型。
2. 成员函数修饰符（比如 const 或 volatile）需要标示出目标，例如'func(int) const' 和 'fun(int)'是不同的成员函数。
3. JavaDoc 兼容将“#”替换成“::”这一形式上的改变。
4. 在一个类文档中包含一个 foo 成员，建立一个全局变量引用时可用‘:: foo’，而#foo 也将链接到 foo 成员。

非重载成员的参数列表可能会被忽略。

如果一个函数被重载，却未指定一个匹配的参数列表（如上述的情况 2 或 6 下），那么只能与这些重载成员中的某一个建立链接。

类空间中的成员函数（在情况 4 或 7 下）有可能被忽略，如果：
1. 指向隶属于同一个类的文档化成员，其实在文档块中已经包含了这种关系。
2. 类的文档块中已经包含有成员函数。

链接到变量，类型转换，枚举类型，枚举值和定义

和可以使用上一节相同的方法来链接这些元素，为了文档的可读性，推荐只用上述情况 3 和 7。
例子：

```
/*! \file autolink.cpp
  Testing automatic link generation.


  A link to a member of the Test class: Test::member,
```

```
  More specific links to the each of the overloaded members:
  Test::member(int) and Test#member(int,int)

  A link to a protected member variable of Test: Test#var,

  A link to the global enumeration type #GlobEnum.

  A link to the define #ABS(x).

  A link to the destructor of the Test class: Test::~Test,

  A link to the typedef ::B.

  A link to the enumeration type Test::EType

  A link to some enumeration values Test::Val1 and ::GVal2
*/

/*!
  Since this documentation block belongs to the class Test no link to
  Test is generated.

  Two ways to link to a constructor are: #Test and Test().

  Links to the destructor are: #~Test and ~Test().

  A link to a member in this class: member().

  More specific links to the each of the overloaded members:
  member(int) and member(int,int).

  A link to the variable #var.

  A link to the global typedef ::B.

  A link to the global enumeration type #GlobEnum.
```

```
  A link to the define ABS(x).

  A link to a variable \link #var using another text\endlink as a link.

  A link to the enumeration type #EType.

  A link to some enumeration values: \link Test::Val1 Val1 \endlink and ::GVal1.

  And last but not least a link to a file: autolink.cpp.

  \sa Inside a see also section any word is checked, so EType,
      Val1, GVal1, ~Test and member will be replaced by links in HTML.
*/

class Test
{
  public:
    Test();               //!< constructor
   ~Test();               //!< destructor
    void member(int);     /**< A member function. Details. */
    void member(int,int); /**< An overloaded member function. Details */

    /** An enum type. More details */
    enum EType {
      Val1,               /**< enum value 1 */
      Val2                /**< enum value 2 */
    };

  protected:
    int var;              /**< A member variable */
};

/*! details. */
Test::Test() { }

/*! details. */
```

```
Test::~Test() { }
```

```
/*! A global variable. */
int globVar;

/*! A global enum. */
enum GlobEnum {
                GVal1,    /*!< global enum value 1 */
                GVal2     /*!< global enum value 2 */
              };

/*!
 *  A macro definition.
 */
#define ABS(x) (((x)>0)?(x):-(x))

typedef Test B;

/*! \fn typedef Test B
 *  A type definition.
 */
```

http://www.stack.nl/~dimitri/doxygen/examples/autolink/html/index.html

类型转换

类型转换包含类，结构和联合，比如：

```
typedef struct StructName TypeName
```

创建一个 StructName 的假名，在建立一个到 StructName 的链接时，既可以链接到 StructName 自身也可以链接到 TypeName。
例子：

```
/*! \file restypedef.cpp
 * An example of resolving typedefs.
 */
```

```
/*! \struct CoordStruct
 * A coordinate pair.
 */
struct CoordStruct
{
  /*! The x coordinate */
  float x;
  /*! The y coordinate */
  float y;
};

/*! Creates a type name for CoordStruct */
typedef CoordStruct Coord;

/*!
 * This function returns the addition of \a c1 and \a c2, i.e:
 * (c1.x+c2.x,c1.y+c2.y)
 */
Coord add(Coord c1,Coord c2)
{
}
```

http://www.stack.nl/~dimitri/doxygen/examples/restypedef/html/restypedef_8cpp.html

输 出 格 式

Doxygen 直接支持以下的输出格式：

HTML

如果 GENERATE_HTML 设定为 YES，将生成 HTML 格式。

LaTeX

如果 GENERATE_LATEX 设定为 YES，将生成 LaTeX 格式。

Man 页面

如果 GENERATE_MAN 设定为 YES，将生成 Man 格式。

RTF

如果 GENERATE_RTF 设定为 YES，将生成 RTF 格式。

注意 RTF 输出可能只可以使用微软 Word97 查阅，如果你成功使用在其他软件中，请告诉我。

XML

如果 GENERATE_XML 设定为 YES，将生成 XML 格式。

Qt 帮助工程（.qhp）

如果 GENERATE_QHP 设定为 YES，将生成 qhp 格式。

以下是 doxygen 间接支持的输出格式：

已编译 HTML 帮助（又称唔 window98 帮助文件）

使用微软 HTML Help workshop 转换 HTML 输出来生成。

Qt 压缩帮助文件（.qch）

使用 Qt 工具 qhelpgenerator 转换 HTML 输出来生成。

PostScript

在 LaTeX 输出目录下运行 make ps 命令生成，为得到最好的效果将 PDF_HYPERLINKS 设为 NO。

PDF

在 LaTeX 输出目录下运行 make pdf 命令生成，要提高 pdf 输出的效果，通常需设定 USE_PDFLATEX 为 YES 使能 pdflatex，为了在 pdf 文件中使用超链接，需要设定 PDF_HYPERLINKS 设为 YES。

搜　　索

Doxygen 能为你的源码建立多种索引，使得你更容易浏览和查询你需要的内容。也可以通过关键字搜索你需要的东西，而不是翻阅它。

HTML 浏览器默认情况下，不具备在多个页面之间进行搜索的能力，但是借助 doxygen 或者外部工具实现这个特性。

Doxygen 有 6 种不同的搜索方法可添加到 HTML 的输出中，当然每一种方法都有其的优缺点。

1. 客户端侧搜索

使能搜索最容易的方法，有效内建的客户侧的搜索引擎。此引擎只能在客户浏览器中使用 JavaScript 和 DHTML 执行，并且它工作时不需要其他附加的工具。

可设定 SEARCHENGINE 为 YES 有效之，确认 SERVER_BASED_SEARCH 已设为 NO。]

此类搜索方法的一个额外优点,提供了智能搜索 live searching,即提交的搜索结果最大限度地符合你的输入。

这类方法的缺陷:它只能搜索符号的局限,它无法提供文本的搜索能力,它缺乏对大型项目的适应能力(搜索的速度变得很慢)。

2. 服务端侧搜索

如果你计划在一个网页服务器上放置 HTML 文档,并且服务器有能力处理 PHP 代码,那么你就可以使用 doxygen 内建的服务端侧的搜索引擎。

设定 SEARCHENGINE 和 SERVER_BASED_SEARCH 为 YES。

相对于客户端侧的优势，在于提供了文本搜索和对大型项目的良好支持。而不利之处则是它无法工作在本地（无法使用 file:// 这样一个 URL），它无法提供智能的搜索能力。

3. windows 编译 HTML 帮助文件

如果在 windows 下运行 doxygen，你可为输出的 HTML 文档创建一个已编译的 HTML 帮助文件（.chm），它是一个包含所有 HTML 文件的单个文件，并且还包括了一个搜索引擎。这许多平台上都有该格式的查看器，windwos 甚至将它作为默认的系统功能。

设定 GENERATE_HTMLHELP 为 YES，可是 doxygen 编译 HTML 帮助文件，但是你需要在 HHC_LOCATION 指定 HTML 编译器（hhc.exe）的路径，以及在 CHM_FILE 指定 chm 的文件名。

此类方法的优点在于单个文件易于发行，也提供文本搜索。
不利之处，只能在 windows 下编译 chm 文件且需要微软不太积极支持的 HTML 编译器，尽管此工具在大多数人手中都工作良好，但是它有时可能因为一些不明的原因而出现意外（如何才是典型应用 how typical）。

4. Mac OS X Doc Sets

如果你是在 Mac OS X 10.5 或更高的版本下运行 doxygen，可为 HTML 输出文档创建一个 doc set，它会在一个预编译搜索索引的基础上，用一个特殊结构来包含 HTML 文件，并将其放置到一个单独的目录下，doc set 能够嵌入到 Xcode 中（Apple 支持的集成开发环境）。

设置 GENERATE_DOCSET 为 YES 使能 doc sets 的创建，你可能需要设置其他一些与 doc set 相关的参数。当 doxygen 运行结束之后，你将在 HTML 输出目录找到一个 Makefile，运行 make install 编译和安装 doc set，参看 http://developer.apple.com/tools/creatingdocsetswithdoxygen.html 获得更多的信息。

此种模式的优点在于它已经完美地集成到 Xcode 开发环境中，允许你在编辑器中点击一个标识符，跳转到适合的文档章节中。

不利之处在于它只能工作在 MacOSX 下，并与 Xcode 组合在一起。

5.Qt 压缩帮助文件

如果你希望将其安装到 Qt 应用框架或在它之下进行开发，你需要获得一个 Qt assistant，这个 Qt 压缩帮助文件（.qch）的查看器。

设置 GENERATE_QHP 为 YES，你需要配置其他与 Qt 相关的选项，比如 QHP_NAMESPACE, QHG_LOCATION,

QHP_VIRTUAL_FOLDER，查阅 http://doc.trolltech.com/qq/qq28-qthelp.html#htmlfilesandhelpprojects 获得更多的信息。

Qt 压缩帮助文件完全可以媲美 chm，它的额外优点是编译 QCH 文件并不仅仅局限于 windows。

不利之处在于它需要 Qt4.5（或者更高）的版本支持，或为文档配备 Qt help assistant 工具，事实上这有点复杂，因为当前 Qt help assistant 还不是一个独立发布的程序包。

6. Eclipse 帮助插件

如果你使用 eclips，你可以将 doxygen 生成的文档嵌入其内，如同一个帮助插件，它将会出现在帮助浏览器中，看上去像是一个主题，并能从帮助菜单的帮助索引 Help contents 中开始，当你使用关键字进行首次搜索时，Eclipse 将为文档生成一个搜索索引。

设置 GENERATE_ECLIPSE_HELP 为 Yes 使能帮助插件，使用 ECLIPSE_DOC_ID 选项能为你的项目定义一个 uid。例如：

```
GENERATE_ECLIPSE_HELP = YES
ECLIPSE_DOC_ID = com.yourcompany.yourproject
```

然后在 eclipse 的 plugin 目录下创建 com.yourcompany.yourproject 目录（可使用 ECLIPSE_DOC_ID 作为同一名称），在 doxygen 完全复制帮助文件输出目录到 com.yourcompany.yourproject 目录后，重启 eclipse 使其找到新的插件。

Eclipse 帮助插件支持一些简单功能如同 Qt 压缩文件和 CHM 文件一样，但是必须在 eclipse 被安装和运行之后才能使用。

自 定 义 输 出

Doxygen 支持多种自定义等级。第一节中将讨论，如果你要对输出文档的效果进行轻度微调，该做些什么。下一节将展示如何在一个页面中重新排序和隐藏某些信息，最后一节将展示如何基于 XML 输出生成你想得到任何信息。

轻度微调

你能创建一个不同的 CSS cascading style sheet（http://www.w3schools.com/css/default.asp），对 HTML 输出进行简单微调，比如字体大小或颜色，页边距，其他观感上的效果。你也能让 doxygen 为每一个 HTML 页面生成一个自定义页头和页脚，比如包含一个 logo 或将 doxygen 输出合并到页面的空余部分。

按以下的命令先运行 doxygen：

```
doxygen -w html header.html footer.html customdoxygen.css
```

这将创建 3 个文件：

- header.html 是一个 doxygen 启动一个 HTML 页面所使用的页面片段，注意此页面片段末尾的正文标记中，包含了来自$word 的几个命令，在页面运行时将被 doxygen 覆盖掉。
- footer.html 是一个 doxygen 终止一个 HTML 页面所使用的页面片段，这里也能使用特殊命名，这个文件包含了 www.doxygen.org 的链接和正文与 HTML 的终止标记。
- customdoxygen.css 是 doxygen 使用的默认 CSS。

你需要编辑 doxygen 配置文件中与这些文件的关联。

```
HTML_HEADER       = header.html
HTML_FOOTER       = footer.html
HTML_STYLESHEET   = customdoxygen.css
```

查阅 HTML_HEADER （http://www.stack.nl/~dimitri/doxygen/config.html#cfg_html_header）标记的文档，获得更多关于元 meta 命令的信息

注意：
不要在 HTML 输出目录放置 style sheet，因为它将被看做一个源文件且会被 doxygen 复制。

如果你在一个自定的页头中使用了图片和其他外部链接，需要靠你自己确认最终的文件是否被加入到 HTML 输出目录中，比如写一个能使 doxygen 复制图片到输出目录的脚本。

改变页面的布局

在某些情况下，你可能需要一种改变输出结构的方法。一个不同的 style sheet 或自定义的页头和页脚，对于此些情况没有任何帮助。

Doxygen 提供了一个布局文件的解决方案，它能被修改以及被 doxygen 用于控制页面信息的显示和如何显示这些信息，这个布局文件是一个 XML 文件。

使用以下命令可以生成一个默认的布局文件:

```
doxygen -l
```

布局文件名可以被任意指定的，默认文件名为 DoxygenLayout.xml。

下一步则是在配置文件设定布局文件

```
LAYOUT_FILE = DoxygenLayout.xml
```

编辑布局文件来改变你需要得到的修正。

布局文件的整体结构如下：

```
<doxygenlayout version="1.0">
  <navindex>
    ...
  </navindex>
  <class>
    ...
  </class>
  <namespace>
    ...
  </namespace>
  <file>
    ...
  </file>
  <group>
```

```
    ...
  </group>
  <directory>
    ...
  </directory>
</doxygenlayout>
```

DoxygenLayout.xml 的根标记是一个名为 version 的属性，用于未来的不会向后兼容的扩展变化。

第一节是闭合的 navindex 标记，描述了在任何一个 HTML 页面都会显示的导航标签的配置，每一个 tab 都是 XML 文件中一个 tab 标记。

你可以在 visible 属性将 tab 设为 no，以隐藏这些 tab，你也可在 title 属性中为 tab 指定一个特殊的标题，以覆盖掉默认的，如果 tab 的标题域是一串空白（默认），那么 doxygen 将为其设定一个合适的标题，你也能在 XML 文件的 navindex 段落中移动 tab 进行重新排序，甚至改变页面树的结构。任何状态下不要改变 type 的值，只有一个种固定的 type 设置能被支持，每一个链接的描述都将指向一个特殊的索引。

在 navindex 之后小节中，将描述 doxygen 所生成的不同页面的布局

- 类一节描述文档化类，结构，联合和接口所生成的全部页面的布局
- 名字空间一节描述文档化名字空间所生成的全部页面的布局（也可以是 Java 的包）
- 文件一节描述文档化文件所生成的全部页面的布局
- 组一节描述文档化 zu 所生成的全部页面的布局（或者模块）
- 目录一节描述文档化目录所生成的全部页面的布局

在以上小节中任意一个内所有的 XML 标记只会出现一部分。有些页面能出现全部类型，其他则是出现某些特定的类型，doxygen 在 XML 文件中将对已出现的类型进行列表。

有些标记有一个 visible 的属性，用以在生成的输出文档中隐藏这一片段。通过设定 visible 为 no，你也能通过在它们名字中加入一个美元符号的前缀，使用配置选项中的这个值来确定标记是否可见，例子：

```
  ...
  <includes visible="$SHOW_INCLUDE_FILES"/>
  ...
```

这是为向后的兼容性准备的。注意，对于 doxygen 来说，visible 属性只是一个提示，如果 visible 设为 YES 但是那一节并无相关的信息需要显示，那么它将被忽略。（比如只生成非空的小节）

许多标记有一个标题的属性，它可自定义标题如同一个块的头部。

警告：
任何时候你不要使用从布局文件中删除标记的方法来隐藏信息，那样做的后果是在生成输出文档中的链接将无效。

以下是每个页面中可能出现的常用标记：

briefdescription
是一个页面的简短描述。
detaileddescription
是一个页面的详细描述。
authorsection
是一个页面编写者的小节（只能用于 man 页面）。
memberdecl
是一个页面所有成员的快速概览（成员声明），此标记有一个子标记，包含了成员类型的列表，列表的信息可能不详细，但子标记的名字将会是一个很好的获取成员类型需求的指示。
memberdef
是一个页面中详细的成员列表（成员定义），象 memberdecl 标记，也可能有一定数量的子标记。

以下是类页面的特殊标记：

includes
包含头文件，用于获取类定义。
inheritancegraph
包含类的继承关系，注意，CLASS_DIAGRAM 选项用于确定继承关系是一个基类和派生类的列表，还是一个继承图。
collaborationgraph
包含一个类的协作图。
allmemberslink
包含类的全部成员的链接。
usedfiles
包含了在文档中使用过该类的文件列表。

以下是文件页面的特殊标记：

includes
包含文件中#include 语句的列表。
includegraph
包含文件的包容依赖图。
includedbygraph
包含了文件所涉及的依赖图。
sourcelink
包含文件源码的链接

组页面中有一个特殊标记 groupgraph，它包含了组之间的依赖图。

同样目录页面也有一个特殊标记 directorygraph，它包含了基于目录中文件的#include 的关系，目录之间的依赖图。

使用 XML 输出

如果上述的两种方法还不能提供足够的灵活性，你可以使用 XML 的输出来建立你所期望的文本输出，它需要设定 GENERATE_XML 为 YES。

XML 输出包含了一个名为 index.xml 的索引文件，它列出了其他 XML 文件中所有条目的细节与引用。此索引的结构又是由一个架构文件 index.xsd 来描述的。所有其他的 XML 文件的描述可在另一个名为 compound.xsd 的架构文件中找到。如果你倾向于使用单个大型的 XML 文件，可用 combine.xslt 的 XSLT 文件将索引和其他文件组合字在一起。

你能够使用任何 XML 解析器来解析这个文件，或者使用 doxygen 发行版本中 addon/doxmlparser 目录下所找到解析器。打开 addon/doxmlparser/include/doxmlintf.h 查阅解析器的接口，打开 addon/doxmlparser/example 查看目录下的例子。

使用 doxmlparser 的优点是，只在内存中读出索引文件，并通过索引的导航，装载你选定的 XML 文件，因为从一个巨型项目中读出所有的 XML 文件，如同是读出一个大型 DOM 树，都是不适合全部载入到内存中。

自 定 义 命 令

Doxygen 支持数量庞大的特殊命令，XML 命令，HTML 命令，可在一个注释块中增强和结构化文档。如果你因为某些原因需要定义一个新的命令，可以使用一个 alias 的定义符。

alias 的功能需要在配置文件中设定 ALIASES 标记使能。

简单的 alias

一个简单的替换操作是 alias 最基本的功能。

```
name=value
```

定义 alias 的例子：

```
ALIASES += sideeffect="\par Side Effects:\n"
```

它允许你在文档中放置\sideeffect (或者 @sideeffect)命令，结果将放在一个标题为 Side Effects:的用户定义的段落中。

注意，你可以在一个 alias 的双引号之间放入\n，来插入一个新行。也能够对已存在的特殊命令依照你自己的意愿重新定义，比如\xrefitem 的一些命令可用与 alias 的组合。

Alias 的参数

Alias 可带一个或多个参数，在 alias 定义的大括号之间你可以指定一定数量的参数，并在定义的参数值设定部分，你能放置\x 标记，此处 x 表示参数号并从 1 开始。

这有一个带单个参数的 alias 定义：

```
ALIASES += 1{1}="\ref \1"
```

在定义之后，你可插入以下的一个注释块

```
/** See \l{SomeClass} for more information. */
```

换一种写法也能表示相同的意义

```
/** See \ref SomeClass for more information. */
```

注意，你可以使用一个带有多个参数的 alias 版本覆盖掉原来的，比如

```
ALIASES += l{1}="\ref \1"
ALIASES += l{2}="\ref \1 \"\2\""
```

注意，在引号内反斜杠之后的字符是能被转义。

以下是新版 alias 的注释块

```
/** See \l{SomeClass,Some Text} for more information. */
```

以上的注释块可被扩展成

```
/** See \ref SomeClass "Some Text" for more information. */
```

如前所示带有单个参数的命令将会一直工作。where the command with a single argument would still work as shown before.

Alias 也能出现其他 alias 的文本行中，例如一个新命令\reminder 也能被包容，如同在\xreflist 中使用\xrefitem 命令，如下所示：

```
ALIASES += xreflist{3}="\xrefitem \1 \"\2\" \"\3\" " \
ALIASES += reminder="\xreflist{reminders,Reminder,Reminders}" \
```

注意，如果 alias 中有多于一个的参数，可以使用一个逗号作为分隔符，如果你希望在命令中放置一个逗号，那么需要使用一个反斜杠将其转义，如：

```
\l{SomeClass,Some text\, with an escaped comma}
```

\1 可获得 alias 的定义。

嵌套自定义命令

你可以使用命令来作为 alias 的参数，当然也包括使用 alias 定义的命令。

以下是 alias 定义的一个例子

```
ALIASES += Bold{1}="<b>\1</b>"
ALIASES += Emph{1}="<em>\1</em>"
```

能插入一个注释块：

```
/** This is a \Bold{bold \Emph{and} Emphasized} text fragment. */
```

并将其扩展成：

```
/** This is a <b>bold <em>and</em> Emphasized</b> text fragment. */
```

链 接 外 部 文 档

如果你的项目依赖于外部库或是工具，由于某些原因，每一次运行 doxygen 时都无法包含所有的源文件。

磁盘空间：
某些与项目有关联的文档有可能在 doxygen 的输出目录之外，例如是网页。你可以希望链接这些页面，放置到你本地输出目录中生成的文档内。

编译速率：
在你自己的项目中包含有一个需频繁更新的其他项目，让 doxygen 去解析这些外部项目的源码，除非它们没有更改，否则需要不停地重新解析覆盖，紧接着再一次解析覆盖，如此运作的意义不大。

内存：
对于一个巨型代码树，让 doxygen 解析所有的源码，操作上固然很简单，但是需要很多系统内存。可将源码分成若干个“包”，doxygen 只需要解析其中一个包，当其他的包与其有关联时，建立一个外部链接即可，这样只需要很少的内存就可工作。

可用性：
有些被 doxygen 文档化的项目，并没有包含源码在其中。

版权问题：
如果外部包和文档的版权都是属于某位作者，并且此包有着非常优异的表现和广泛的应用，那么你与其引用它倒不如在你的项目文档中包含一个此包的副本。一旦作者中止发布，或者需要遵守某些授权条件时，你就可以不去理会这些条件限制，继续使用你先前的副本。

如果应用上的任何问题，你都能使用 doxygen 的标记文件的机制来处理。一个标记文件是基于外部源码的一个简明的描述集合。Doxygen 能够创建并读取标记文件。

为你的项目创建一个标记文件，只需要在配置文件的 GENERATE_TAGFILE 选项之后放置标记文件的名称。

在你自己的项目中组合一个或多个外部项目的输出，需要在配置文件的 TAGFILES 选项之后指定标记文件的名称。

一个标记文件并不包含外部文档的本地信息，它可能只是一个目录或者 URL，所以当你包含一个标记文件时，你将不得不指定外部文档的本地位置，有两种方法可以做到这一点：

在配置时：
立即在 TAGFILES 配置选项之后分配一个输出的本地位置给标记文件，如果你使用的是一个相对路径，那么它一定是相对于你项目所生成的 HTML 输出目录。

编译之后：
如果你没有为标记文件分配一个本地位置，doxygen 将为所有外部的 HTML 引用创建虚拟链接，也将在 HTML 输出目录中生成一个 installdox 的 perl 脚本，这个脚本运行之后，全部已生成的 HTML 文件的虚拟链接，将被实链接所代替。

例子：
假定你有一个 proj 的项目，使用了两个外部项目 ext1 和 ext2，目录结构如下：

```
<root>
  +- proj
  |   +- html               HTML output directory for proj
  |   +- src                sources for proj
  |   |- proj.cpp
  +- ext1
  |   +- html               HTML output directory for ext1
  |   |- ext1.tag           tag file for ext1
  +- ext2
  |   +- html               HTML output directory for ext2
  |   |- ext2.tag           tag file for ext2
  |- proj.cfg               doxygen configuration file for proj
  |- ext1.cfg               doxygen configuration file for ext1
  |- ext2.cfg               doxygen configuration file for ext2
```

配置文件中的相关部分如下：

proj.cfg：

```
OUTPUT_DIRECTORY  = proj
INPUT             = proj/src
TAGFILES          = ext1/ext1.tag=../../ext1/html \
                    ext2/ext2.tag=../../ext2/html
```

ext1.cfg:

```
OUTPUT_DIRECTORY  = ext1
GENERATE_TAGFILE  = ext1/ext1.tag
```

ext2.cfg:

```
OUTPUT_DIRECTORY  = ext2
GENERATE_TAGFILE  = ext2/ext2.tag
```

在一些情况下（希望有例外），你可能有 doxygen 已生成好的文档，但这些源码并不一定有一个标记文件，这种情况下你能使用 doxytag 工具从已生成 HTML 文件中提取一个标记文件。另一类情况，你也可以为 Qt 文档创建一个标记文件。

Doxytag 工具依赖于已生成输出的特定结构和 doxygen 所生成一些特殊标示。这类提取方法比较容易出错以及不够健壮，如果无其他的选择时，我建议你还是只使用这种方法，在将来 doxytag 可能会被废弃。

FAQ

1. 如果在 HTML 的索引页面中获取信息？

你可以插入\mainpage 命令到一个注释块中，如同：

```
/*! \mainpage My Personal Index Page
*
* \section intro_sec Introduction
*
* This is the introduction.
*
* \section install_sec Installation
*
* \subsection step1 Step 1: Opening the box
*
* etc...
*/
```

2. 求助，你的类/文件/名字空间中的部分或是全部成员无法被文档化？

请看以下：

1. 是不是在你的类/文件/名字空间文档化的过程中出现该问题？如果不是，它们是不会从源码被提取出来，除非 EXTRACT_ALL 被设为 YES。

2. 这些成员是否为私有类型，如果是，你必须设置 EXTRACT_PRIVATE 为 YES，使得它们出现在文档中。

3. 你的类中是否有函数宏没有使用分号结束（例如 MY_MACRO()）？如果是，那么你指示 doxygen 的预处理将其删除。

以下有一个配置文件中的典型设定：

```
ENABLE_PREPROCESSING   = YES
MACRO_EXPANSION        = YES
EXPAND_ONLY_PREDEF     = YES
PREDEFINED             = MY_MACRO()=
```

阅读“预处理”节的指南来获取更多的信息。

3. 当我设置 EXTRACT_ALL 为 NO 时，我的所有函数都没有出现在文档中？

为了文档化全局函数，变量，枚举，类型转换和定义，你必须文档化包含上述成员的文件，可在本地使用一个包含\file（或 @file)命令的注释块。

另外，你可以使用\ingroup 命令，放置所有的成员到一个组（或模块）中，然后用包含\defgroup 命令的注释块来文档化该组。

如果是一个名字空间中的函数或是成员函数，那么你必须要文档化函数所在的类或是名字空间。

4. 如何使得 doxygen 忽略一些代码片段？

这有一个最新和简便的方法，在代码片段的起始处放置一个包含\cond 命令的注释块，并在片段的末尾处放置一个\endcond 命令的注释块，即可忽略掉两个命令之间的代码，当然它必须是在同一个文件中。

你也可以使用 doxygen 的预处理来实现，如果你放置

```
#ifndef DOXYGEN_SHOULD_SKIP_THIS

/* code that must be skipped by Doxygen */

#endif /* DOXYGEN_SHOULD_SKIP_THIS */
```

在此块中的代码将被隐藏。在配置文件放置以下内容，那么所有标记为 DOXYGEN_SHOULD_SKIP_THIS 的块都将被跳过，需要设置 PREPROCESSING = YES。

```
PREDEFINED = DOXYGEN_SHOULD_SKIP_THIS
```

5. 在类文档中如何改变#include 之后的内容？

大多数情况下，你可以使用 STRIP_FROM_INC_PATH 去除用户定义的路径。

你也可以按以下的例子来文件化你的类

```
/*! \class MyClassName include.h path/include.h
*
*  Docs for MyClassName
*/
```

它将使得 doxygen 在 MyClassName 类的文档中，放置#include <path/include.h>，不理会 MyClassName 所包含的实际头文件名字的定义。

如果你需要 doxygen 显示头文件，可使用双引号来替换尖括号：

```
/*! \class MyClassName myhdr.h "path/myhdr.h"
*
*  Docs for MyClassName
*/
```

6. 如何使用压缩 HTML 文档中已合并的标记文件？

如果你希望在一个压缩 HTML 文件 a.chm 中创建一个引用，链接到另一个压缩 HTML 文件 b.chm，那么此链接必须使用以下的格式：

```
<a href="b.chm::/file.html">
```

可惜这种方法只能工作在两个文件处于同一目录下。

或者使用另一种方法，为项目生成一个 index.chm 的索引文件，并保证它在目录中命名的唯一性，否则可将其改名，然后放置所有与之关联的.chm 文件到同一目录中。

假定你有一个项目 a，使用 b.tag 文件，对项目 b 进行引用，你需重命名项目 a 的 index.chm，并将此更改同步到 a.chm，以及重命名项目 b 的 index.chm，同样将变化同步到 b.chm。
在项目的配置文件可写入：

```
TAGFILES = b.tag=b.chm::
```

你就可以使用 installdox 来设定链接，如下：

```
installdox -lb.tag@b.chm::
```

7. 如果我不希望在每一个 HTML 页面都放置快速索引，那么我该怎么做？

你可以先设定 DISABLE_INDEX 为 YES 取消快速索引，那么你能在你自己的头文件中，写入你自己的页头和页尾到 HTML_HEADER。

8. 当我只使用我自己的头文件时，为什么所有的 HTML 的输出看起来都不一样了？

你可能是在 doxygen 生成时，忘记包含 doxygen.css 样式表。你可以在 HTML 页面的页头的小节放置以下内容中。

```
<LINK HREF="doxygen.css" REL="stylesheet" TYPE="text/css">
```

9. doxygen 为什么要用到 Qt？

一个非常重要的原因，Qt 是一个抽象的平台，包含大多数完整和窗口的应用，例如 QFile, QFileInfo, QDir, QDate, QTime ， QIODevice 这些类，另一原因是具有功能完善和免费支持的类，比如 QList, QDict, QString, QArray, QTextStream, QRegExp, QXML 等等。

GUI 前端 doxywizard 需要使用 QT，这个异常优秀的 GUI。

10. 我如何从自己的目录树中剔除所有的测试目录？

简单的在配置文件中放置如下的剔除选项：

```
EXCLUDE_PATTERNS = */test/*
```

11. doxygen 可自动从正在运行的文本中创建一个到 MyClass 类的链接，如何阻止它在某些地方放置链接？

放置一个“%”在类名之前，比如：%MyClass。Doxygen 将会删除%符号，并不对该类名生成链接。

12. 如果我喜欢的编程语言是 X，我能否一直使用 doxygen？

你对 doxygen 的理解不太正确。Doxygen 需要理解它所读取到的结构，如果你不想花费一些时间在 doxygen 上，那么这里有几种选择：

○ 如果 X 的语法接近 C 或 C++，那么源码/扫描器的调整大概不会太困难，一个比特位来代表一行，此类语言是支持的。l a bit so the language is supported. Doxygen 能直接支持其他的一些语言。（比如 Java，IDL，C#，PHP）
○ 如果 X 的语法在某些地方不同于 C/C++，那么可以编写输入过滤器来转换 X，使得 doxygen 能够理解。（与此类情况接近的语言有，VB，Object Pascal, Javascript，查阅 http://www.stack.nl/~dimitri/doxygen/download.html #helpers 说明）
○ 如果语法完全不兼容 C/C++，那就需要编写一个 X 的解析器，以及一个类似于语法树的后端，提供给源码/扫描器使用。（也可以使用 src/tagreader.cpp 来读取标记文件）

13. 求助！我得到一个神秘的提示“input buffer overflow, can't enlarge buffer because scanner uses REJECT”？

这个错误的发生，由于 doxygen 的词汇扫描器设置一个规则——在一次提取中匹配的输入字符超过 256k，我在一个巨大的生成文件（大于 256k 行）中遇到过这种问题，此刻内建的预处理器将其转换到一个空文件（大于 256k 的新行）中。另一种情况下它也可能会发生，如果你的代码行数超过 256k。

如果你在运行也出现类似的问题，并且希望我来修正它，请你将触发这个提示的代码片段发送给我，若要解决这个问题，可在你的文件中放置若干换行符，将文件分拆成更小的部分，或使用 EXCLUDE 选项将其排除在输入之外。

14. 当在 latex 目录中运行 make 时，我得到“TeX capacity exceeded”，现在怎么办？

编辑 texmf.cfg 文件，增加各类缓冲区的默认值，然后运行 texconfig init。

15. 在点图中为什么 STL 类的依赖没有显示出来？

Doxygen 无法解析 STL 类，除非 BUILTIN_STL_SUPPORT 选项被打开。

16. 我有一个问题，关于 PHP5 和 windows 下的搜索引擎？

查看 http://www.stack.nl/~dimitri/doxygen/searchengine.html 的解决方法。

17. 可以在命令行中配置 doxygen 吗？

无法通过命令行进行配置，但是 doxygen 可从 stdin 读取，所以你可以使用 pipe 来实现，这有一个例子，告知你如何从命令行中覆盖掉配置文件中的一个选项。（假定是一个 unix 环境）：

```
( cat Doxyfile ; echo "PROJECT_NUMBER=1.0" ) | doxygen -
```

如果指定的多个选项使用了相同的名字，那么 doxygen 将使用最后一个，如果是对于一个已有选项的添加，可以使用+=操作符。

18. doxygen 如何获得它们的名字？

Doxygen 获得它们的名字，来自于生成器和运行中的文档。

```
documentation -> docs -> dox
generator -> gen
```

在 lex 和 yacc 中查看，大多数操作都是从“yy”开始，其实多出的一个“y”是一个失误，可以确定某些动作（正确的声明是 Docs-ee-gen，同样也多出一个“e”）。 so the "y" slipped in and made things pronounceable (the proper pronouncement is Docs-ee-gen, so with a long "e").

19. 开发 doxygen 的原因是什么？

我曾经编写过一个基于 Qt 库（它在 http://qdbttabular.sourceforge.net/下一直都有效，并由 Sven Meyer 维护）的 GUI 部件。当时 Qt 就能生成非常漂亮的文档（使用 Qt 不想发布的内部工具），而我只能手写相同的文档。这对于维护来是一个噩梦，所以我想要一个与 Qt 类似的工具。我发现了 Doc++，但是马上就认识它还不够好（它不支持信号和槽，并且在 Qt 下的观感和体验，无法满足我的要求），所以我开始编写我自己的工具

排　　错

已知的问题：

- 如果你从源码中编译 doxygen 出现问题，请先阅读“**安装**”。
- doxygen 不是一个真正的编译器，它只是一个词汇扫描器，这意味着它无法从你的源码中检测出错误。
- 它不可能对所有的代码片段进行测试，这是非常有可能，甚至 C/C++一些有效特性都不能进行适当的处理。如果你发现类似的情况，请将它发送给我，使得我能改进 doxygen 的解析能力。请你尝试发送一段尽可能短的代码，以帮助我缩小搜索的范围。
- 如果你的代码中存在类，结构或联合使用了相同的名称，doxygen 将无法正常识别，它还不至于导致系统崩溃，但却会保留这些同名类别中的一个，而忽略其他所有的。
- 有些命令作为其他命令的参数时，是无法工作的。例如在一个 HTML 链接（<a href="...">...<a>）中插入其他的命令（包括其他 HTML 命令）则无法工作！分隔命令是一个很重要的例外。The sectioning commands are an important exception
- 在某些情况下，多余的括号可能会造成 doxygen 的理解错误，例如：

  ```
  void f (int);
  ```

  它将被解析为一个函数声明，但是

  ```
  const int (a);
  ```

  这也将被看成一个名为 int 的函数声明，因为这类风格只会被解释成前面的结果，doxygen 无法做到语义分析。如果多余的能检测到，如下

  ```
  int *(a[20]);
  ```

  那么 doxygen 将会删除括号，正确地解析出结果。
- 并不是代码段中所有的命名符，都能通过链接替换（当使用 SOURCE_BROWSER = YES 时）和重载成员的链接，被包含在文档内，链接亦有可能指向了错误的成员，而这些也都保留在每个函数所生成的“引用”列表中。

  由于代码解析器还不够智能，我也会在不久的将来尝试改进，但是即便这些改进都完成了，也不能百分之百地保证每一个链接都准确无误地连接了相应的文档，因为查找到的代码段的上下文中，所能提供的信息可能存在完整性和二义性的问题。

- 一种不可能发生的情况，使用\relates 或 \relatesalso 命令将一个非成员函数 f 插入到 A 类中，而 A 类中已经存在了一个与 f 同名且参数表一模一样的成员函数。

- 目前对于成员的重用还有很大的限制，它只能工作在指定的模板中。

- 所有命令无法正确完成对 RTF 的转换

- Dot 1.86 版本（可能更早的版本也一样）不能生成正确的 map 文件，它将造成 doxygen 所生成的图无法点击。

- 只在 PHP 下，doxygen 需要获取一个函数/方法中，包含的全部 PHP 语句（也就是代码），否则你在解析它时可能会出现问题。

如何获得帮助

Doxygen 的开发将会紧密地跟随你的需求。

如果你尝试过 doxygen 之后，可否让我知道你对于它的一些想法（你会不会忽略了一些特性？），假设你确定无法使用它，请你告诉我为什么。

如何报告一个 bug

在 GNOME 的 bugzilla 数据库中存放已被追踪的所有 bug，在发送一个新的 bug 报告时，首先搜索一下数据库，可能其他人已经发送了相同的 bug 报告（doxygen 会有一个预先的筛选）。如果你确定已经找到一个新的 bug，请在 http://bugzilla.gnome.org/enter_bug.cgi?product=doxygen 报告。

如果你不确定它是不是一个 bug，你可以首先在用户邮件列表 http://sourceforge.net/mail/?group_id=5971 中寻求帮助。（这需要订阅）

如果你发出一个（含混不清）的 bug 描述，那么通常你是无法得到一个有价值的帮助，它只会使得你花费更多的时间来让别人理解你的意图，最坏的情况下你的 bug 报告，可能会给我完全忽略，所以在你 bug 报告中尝试包含以下的信息：

- 你所使用的 doxygen 版本（比如 1.5.3，使用 doxygen –version 命令来获得版本号）
- 你所使用的操作系统的名称和版本号（比如 SUSE Linux 6.4）
- 发送一份配置文件，通常这是一个非常好的主意，但请你使用 doxygen 时加入-s 参数，它可以生成最小的配置文件。（使用 doxygen －s －u [configName]，从已存在配置文件中删除注释）

- 对我来说修正 bug 的一个更原始的方法，就是如果你能附带一个小的例子来描述你所报告的 bug，我会将它复制到我的机器中。请确定这个例子是一份有效的源码（可以编译），并且这个例子真的能够导致 bug 问题的出现（我经常获得一些无法触发真实 bug 的例子!），如果你打算发送多个文件，请使用 zip 或 tar 工具将这些文件打包成单个文件，以方便处理。注意，当报告一个新 bug 时，你将会获得一个机会，在提交初始 bug 描述的文件中附加上你的报告。

你可以（我鼓励你这样做）为一个 bug 添加一个补丁，如果你这样做了，请在 bug 报告的正文中使用 PATCH 这个关键字。

如果你有了一个如何修正已存在 bug 或限制的想法，请在开发者邮件列表 http://sourceforge.net/mail/?group_id=5971 中讨论它，补丁也可以直接发送到 dimitri@stack.nl ，如果你不喜欢通过 bug 追踪和邮件列表的方式发送。

请使用 diff -uN 或包含你修改过文件包，来创建一个补丁。如果你要发送多个文件，请使用 tar 或 zip 打包工具，我只要下载和保存一个文件即可。

第二部分　　用户参考

特　点

- 文档创建只需要很小的开销，具有简约的文档风格，也可以使用特殊命令和 HTML 输出标记，来创建更多丰富的样式或是结构化的文档。
- 跨平台：能工作在 windows 以及大部分类 Unix（包括 Linux 和 MacOSX）平台下。
- 甚至能从无文档的代码中，提供索引，组织，生成可浏览和交叉引用的输出。
- 使用外部工具为源码生成结构化的 XML 输出
- 支持 C/C++，Java，（CORBA 和微软）Java，Python，VHDL，PHP IDL，C#，Objective-C 2.0，以及一些 extent D，Fortran 的源码
- 支持文件，名字空间，包，类，结构，联合，模板，变量，函数，类型转换，枚举及定义的文档。
- 兼容 JavaDoc（1.1），qdoc3（部分），以及 ECMA-334（C#规范）。
- 带有一个 GUI 前端（doxywizard），它能非常容易的编辑选项和运行 doxygen，并且可运行在 windows，Linux，MacOSX 下。
- 在 HTML（可点击图元）和 LaTeX（内嵌的 PostScript 图形）下，自动生成类以及它的协作图表。
- 使用 Graphviz 工具箱中的 dot 工具，生成依赖图，协作图表，调用图，目录结构图，类层级图。
- 允许组合模块的主体，并能创建模块的层次。
- 灵活的注释布局：允许你在头文件（声明之前），源文件（定义之前）放置文本，或是一个单独的文件中。
- 可在保护模式下，生成一个类中所有成员的列表（包含继承成员）。
- 同时支持在线格式（XHTML 和 Unix Man）和离线格式（LaTeX 和 RTF）的输出文档（在某些情况下它们可能无效），所有格式都已经进行过优化，易于读取。
  此外，使用微软的 HTML Help Workshop 可从 HTML 输出生成 HTML 压缩文件，以及从 LaTeX 输出获得 PDF 文档。
- 支持多种第三方的帮助文件格式，包括 HTML 帮助文件，docsets，Qt-Help 和 eclipse 帮助文件。
- 包含一个完整的 C 预处理器，能正确解析预处理条件代码段，以及正确展开所有或部分宏定义。
- 自动检测公有，私有，保护域，并且能良好地支持 Qt 的信号和槽机制，可选择是否提取私有的类成员。
- 自动生成文档化后的类，文件，名字空间，成员的引用，也支持全局函数，全局变量，类型转换，定义，枚举的归档。
- 自动生成基类或超类与继承或重载成员的引用。
- 包含一个快速，分层的搜索引擎，用于在类和成员文档（基于 PHP）中搜索关键字或是字串。
- 包含一个 JavaScript 的智能搜索，用于搜索符号（主要针对小型，中型的项目）。
- 能在文档中输入标准的 HTML 标记，doxygen 能自动将其转换成相同的 LaTeX，RTF 或 man 的副本。
- 允许生成文档与其他（doxygen 文档）项目（或是同一项目中的其他部分），也就是在本地创建的一种非依赖关联的方法。

- 允许加入源代码到文档中，并能自动创建交叉引用。
- 也支持未文档化的类列表，帮助你快速熟悉一个大型代码中的结构和接口，查看执行的细节。
- 自动对源码中的定义创建与文档之间的交叉引用。
- 所有源码段都加入语法高亮，使之易于阅读。
- 允许列出文档中的函数/成员/类的定义。
- 对配置文件添加注释，使得所有选项更易于编辑参数。
- 当未重新生成文档时，文档和搜索引擎能被转移到另一个本地位置或者机器中。
- 支持多种不同字符编码，在生成输出中默认使用 UTF-8。
- Doxygen 能生成一个布局文件，用于编辑和更改每个页面的布局。
- 有超过 100 个配置选项，用于调整输出。
- 能非常容易适应大型项目。

尽管 doxygen 现在能被用于一些项目（doxygen 所支持的语言所编写的）中，起初它是特别为要用到 Qt 工具箱的项目而设计的，我曾经尝试过使得 doxygen 能完全兼容 Qt，所以 doxygen 能读取 Qt 源码中包含的文档，以及创建了一个看上去与 Qt 软件很相似的类浏览器，doxygen 能够理解 Qt 对 C++的扩展，比如信号与槽以及其他很多特殊的标记。

Doxygen 也能自动生成已存在文档的链接，此文档是由 doxygen 创建或者 Qt 非公开的类浏览生成器所创建的，当在一个 Qt 项目中，则意味着你在任何时候都建立成员或类与 Qt 工具箱之间的引用，这个链接将在 Qt 项目文档中被生成，而这些与文档的定位是无任何关联的。This is done independent of where this documentation is located!

Doxygen 用 法

Doxygen 是一个基于命令行的应用程序，运行 doxygen 时附加--help 选项，你将得到一个关于程序用法的简要描述。

所有的选项都包含一个“-”符号，并跟随一个字符和一个或多个参数。

为你的项目生成一个指南，需要以下的几个步骤：

1. 使用特殊文档块来文档化你的源码。（查阅“**特殊文档块**”章节）
2. 调用 doxygen 并使用–p 选项，生成一个配置文件（查阅“**配置**”章节）

```
doxygen -g <config_file>
```

3. 编辑配置文件，使之匹配你的项目，在配置文件中你可以指定输入文件和一些选项信息。
4. 让 doxygen 生成基于配置文件设定的文档。

```
doxygen <config_file>
```

如果你有了一个老版本 doxygen 所生成的配置文件，你可以使用-u 选项将配置文件升级为当前版本。

```
doxygen -u <config_file>
```

原始配置文件中所有配置设定，将被复制新的配置文件，其中一些新加入的选项将使用它们的默认值。注意，在原始配置文件中你所添加的注释将会丢失。

如果你想对输出的观感进行调整，doxygen 允许你在编辑完成之后，生成默认的样式表，文件头和文件尾。

- HTML 输出，你能生成默认文件头（查看“HTML_HEADER”），和默认文件尾（查看“HTML_FOOTER”），和默认的样式表（查看“HTML_SYLESHEET”），使用以下的命令：

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

- LaTex 输出，你能生成 refman.tex 的起始部分（查看“LATEX_HEADER”），和页头包含的样式表（通常是 doxygen.sty），使用：

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

如果你需要非默认的选项（比如要用到 pdflatex），那么要创建一个配置文件，并包含这些选项的正确设定，然后指定这个配置文件作为上述命令的第四个参数。

- RTF 输出，你能生成默认样式表文件（查看 RTF_STYLESHEET_FILE），使用：

```
doxygen -w rtf rtfstyle.cfg
```

注意：

- 如果你不想文档化配置文件中的每一个条目，那么你可以使用-s 选项，也能使用-u 选项进行合并，从一个已存在的配置文件增加或删减文档，如果你发送给我一个配置文件，并作为 bug 报告的一部分，请使用-s 选项。
- 使用 doxygen 读写标准输入输出 stdin/stdout，来替代对文件的读写功能，可使用“-”带文件名。

Doxytag 用 法

Doxytag 是基于命令行的小程序，它能生成标记文件。Doxygen 能使用标记文件生成对外部文档的引用。（例如，未包含在输入文件中的文档）

一个标记文件内包含了外部文档的文件，类和成员的信息。Doxytag 直接从 HTML 文件中提取这些信息，这种方式的优点是，你无需获得源码而是从文档中提取。

如果你没有源码，最好让 doxygen 生成一个标记文件，并将标记文件名放置到 GENERATE_TAGFILE 之后。

Doxytag 的输入包括了一个 HTML 文件集。

重要：
如果你使用标记文件，doxygen 生成的链接中将会包括空链接，那么你需要运行 installdox 脚本将这些空链接更改成有效链接。查阅“**Installdox 用法**”获得更多的信息。看上去使用空链接好像有点多余，其实它是非常有用的，如果你想移动外部文档到另一个本地位置，那么这些文档无需由 doxygen 重新生成，只要运行 installdox 即可。

注意：
因为 HTML 文件都会有一个约定的格式，并且只能用 doxygen 或 Qt 类浏览生成器来生成 HTML 文件，doxytag 也只能读取 HTML 文件，而使用其他方法它们是无法被更改的。

Doxytag 预定一个文档或目录中所包含的全部 HTML 文件的列表，如果列表不存在，doxytag 将读取当前目录中所有扩展名为 html 的文件，如果 doxytag 使用了-t 选项，它将生成一个标记文件。

例子 1：
假定 example.cpp 文件处于 examples 目录下，并且目录中包含了一些未获得源码的组件，幸好组件的分发版本中含有 doxygen 生成的 HTML 文档。

```
/** A Test class.
 *  More details about this class.
 */

class Test
{
  public:
    /** An example member function.
     *  More details about this function.
```

```
     */
    void example();
};

void Test::example() {}

/** \example example_test.cpp
 * This is an example of how to use the Test class.
 * More details about this example.
 */
```

那么你能从组件的 HTML 文件中创建一个标记文件：

```
doxytag -t example.tag example/html
```

在 examples 目录下，最后你能使标记文件成为你代码的一部分，如以下的例子那样：

```
/*! A class that is inherited from the external class Test.
*/

class Tag : public Test
{
  public:
    /*! an overloaded member. */
    void example();
};
```

Doxygen 已使得你自己的文档中包含了外部组件的链接，因为标记文件无法指定文档的本地位置，你还需要运行 doxygen 所生成的 installdox 脚本，指定文档的本地位置。（查阅“installdox 用法”章节来获得更多的信息）

注意,这是一个实用的特性，如果你（或是某人）将外部文档移动到一个不同的目录或是不同的 URL 下，你可以简单地再次运行 installdox 的脚本，所有 HTML 文件的链接将被更新。

Click here for the corresponding HTML documentation that is generated by Doxygen using only the tag file and second piece of code.
（链接为空，不知笔者此处何意，应该是文档没有同步的问题。）

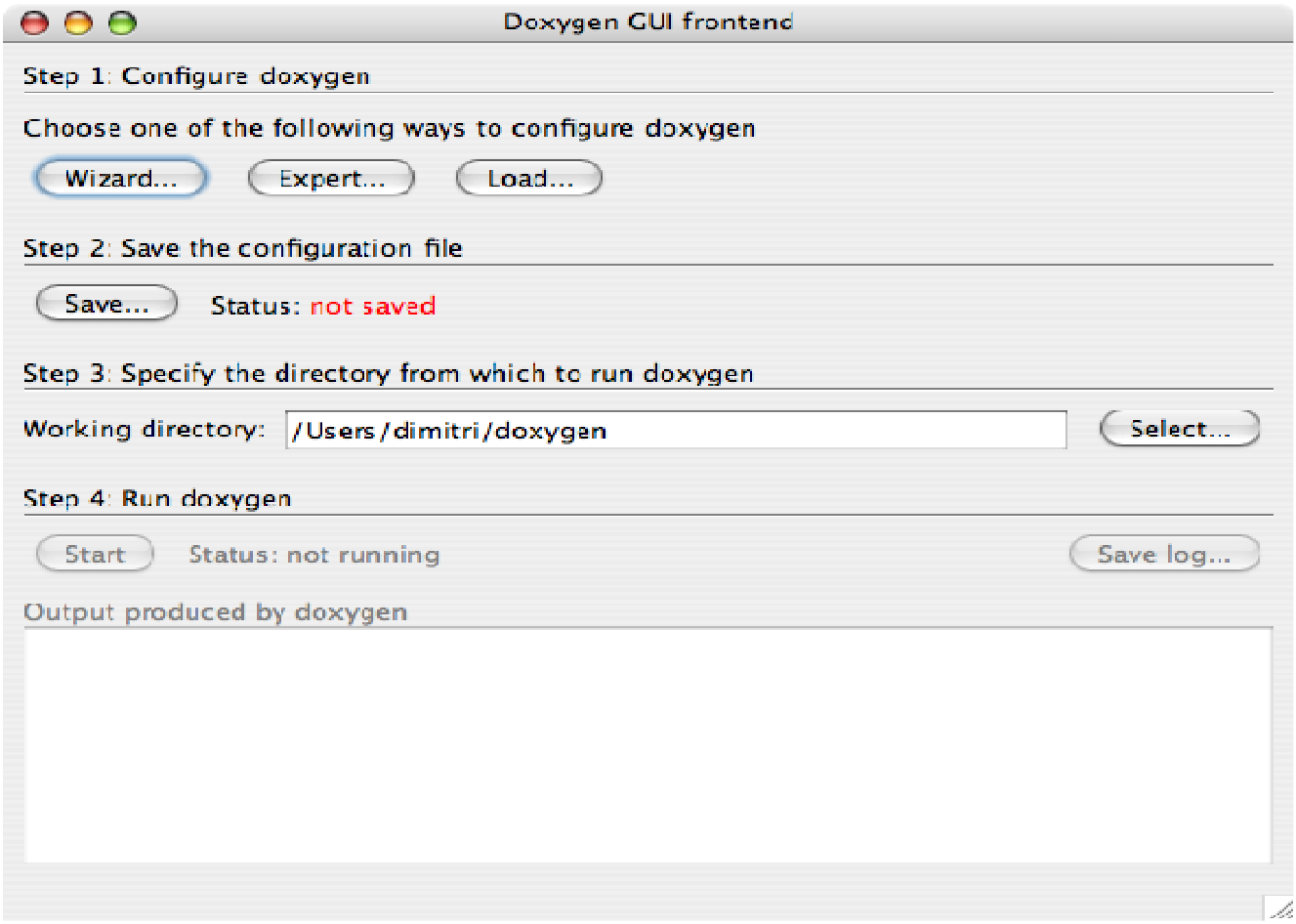
例子 2：
生成一个 Qt 文档的标记文件，可照如下所示：

```
doxytag -t qt.tag $QTDIR/doc/html
```

Doxywizard 用 法

Doxywizard 是一个运行和配置 doxygen 的图形前端。

当你运行 doxywizard 时，将显示的如下的主窗口（实际窗口的观感与操作系统有关）。



主窗口

这个窗口展示了配置和运行 doxygen 所需的若干步骤，第一步，选择一种配置 doxygen 的方法。

向导模式：
点击这个按钮，将快速配置大多数重要的设定，准许其他的选项保有默认值。
专家模式：
点击这个按钮，进入“**全部的配置选项**”
装载模式：
点击这个按钮，将从磁盘中装载一个已存在的配置文件。

注意，你可以在这一行中选择多个按钮，比如先使用向导模式配置 doxygen，然后通过专家模式调整一些设定。

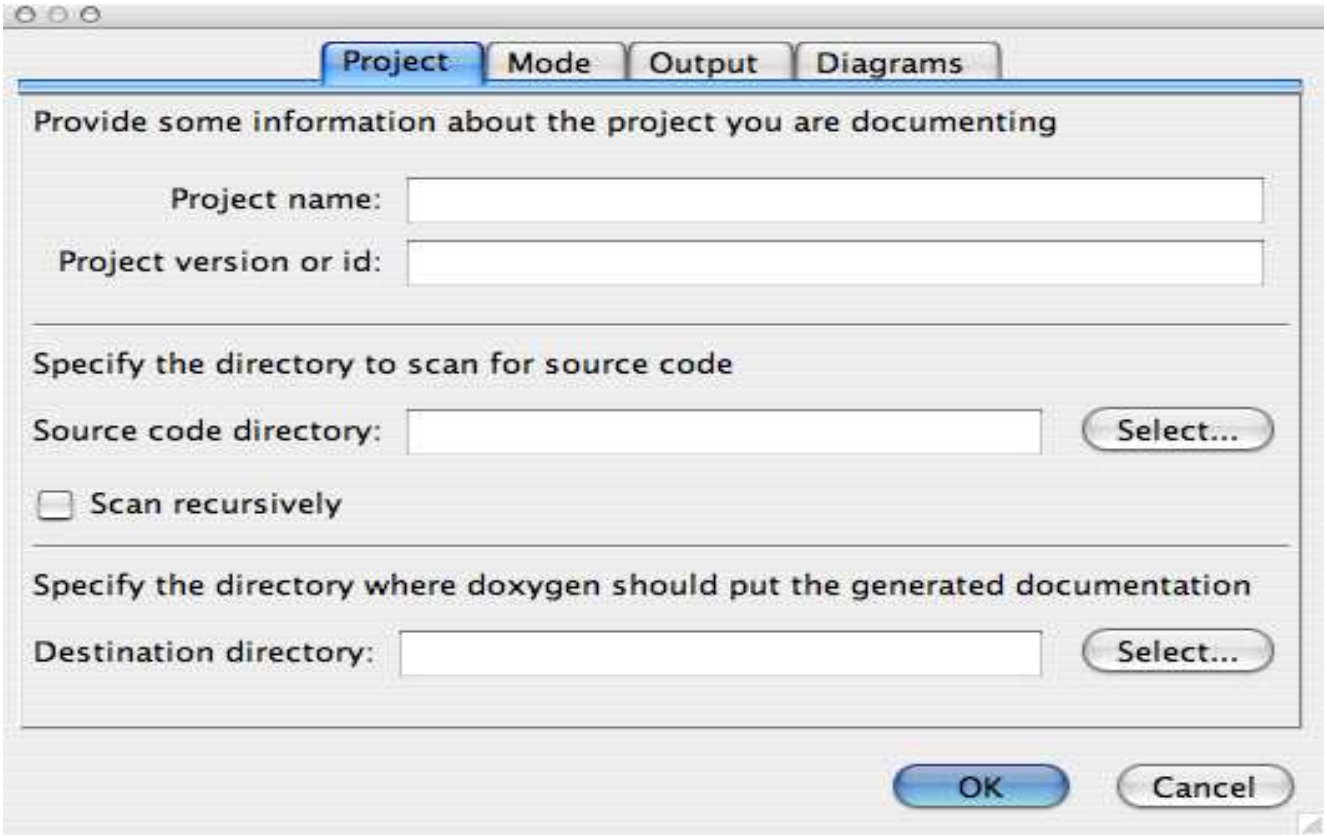
Doxygen 完成配置之后需要将其保存成一个文件放置到磁盘中。第二步，允许 doxygen 使用保存的配置文件，还有一个附带的优点，能够重用配置文件，即使用时间点靠后的同一设定来运行 doxygen。

有一些配置选项会用到相对路径，在下一步可选择 doxygen 的工作目录，一般为代码树的根目录，保证目录被正确填写到其中。This is typically the root of the source tree and will most of the time already be filled in correctly

当配置文件被保存且设定好工作目录之后，你就能在已选定的配置下运行 doxygen，按下 Start 按钮，当你再次点击 Start 按钮，将会终止 doxygen 的运行。捕获到 doxygen 的输出过程将被显示在日志窗口中，当 doxygen 完成处理之后，日志将被保存为一个文本文件。

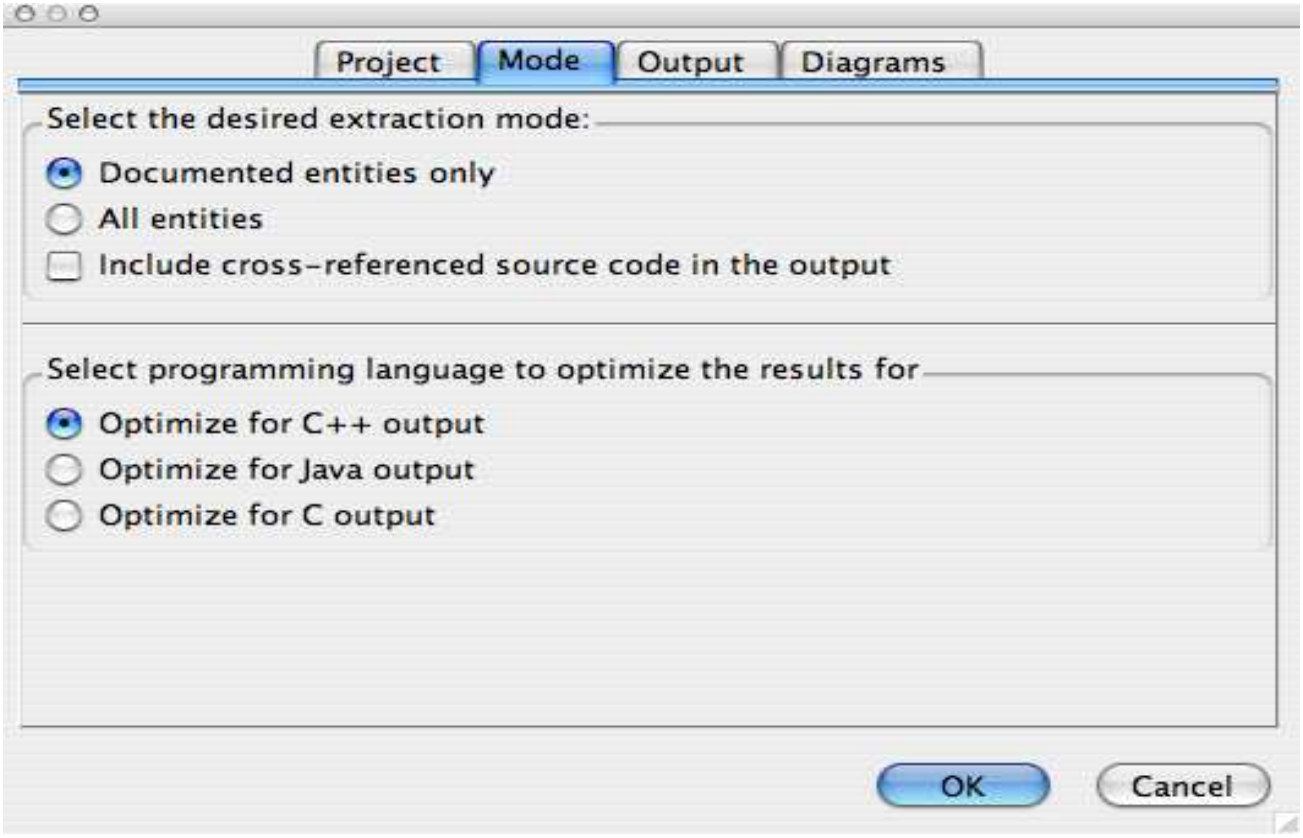
向导对话框

如果你在第一步时选择 Wizard 按钮，就会有一个包含若干选项卡的对话框出现。



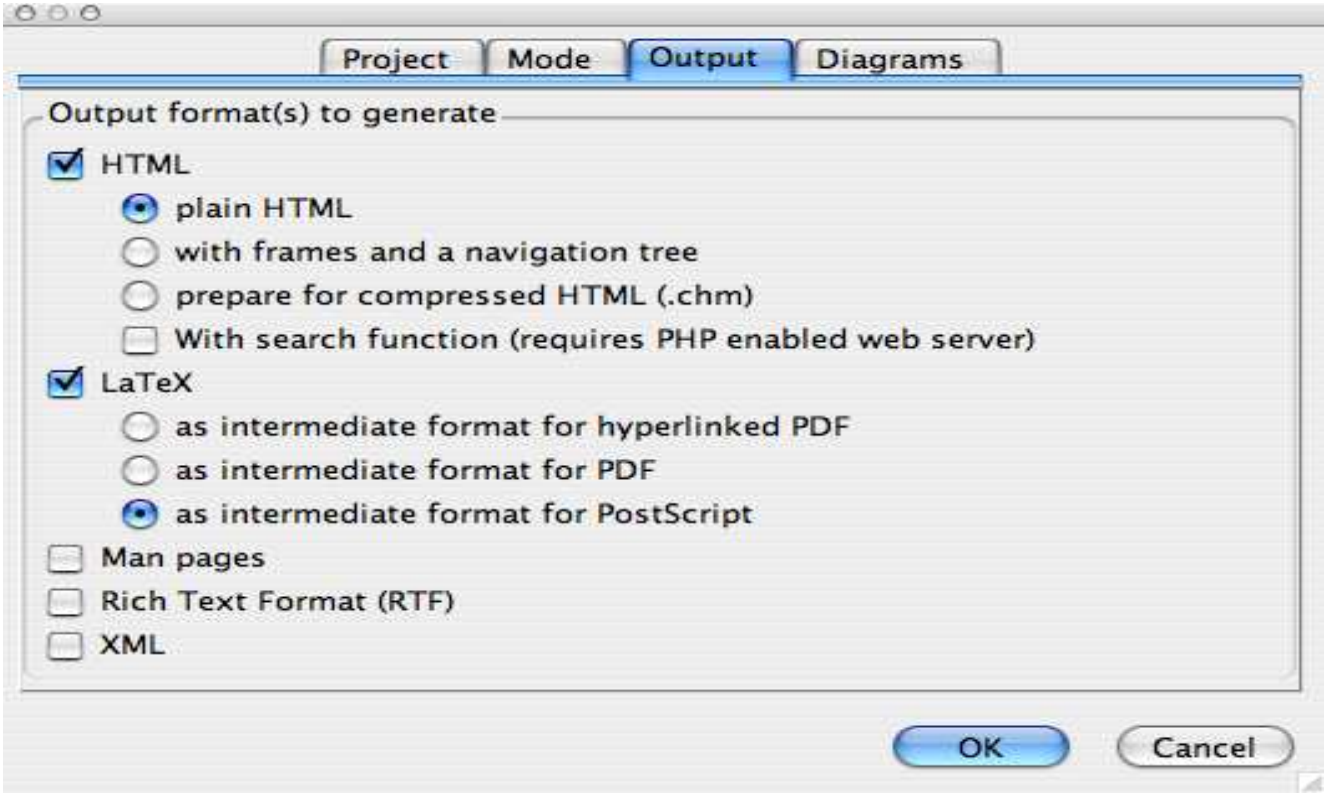
向导对话框：项目设置

上图即为 project 卡全部内容，当 doxygen 完成操作之后，可以在 Destination directory 指定的目录下找到 doxygen 所生成的文档。Doxygen 将会在一个单独的子目录中放置每一种输出格式。



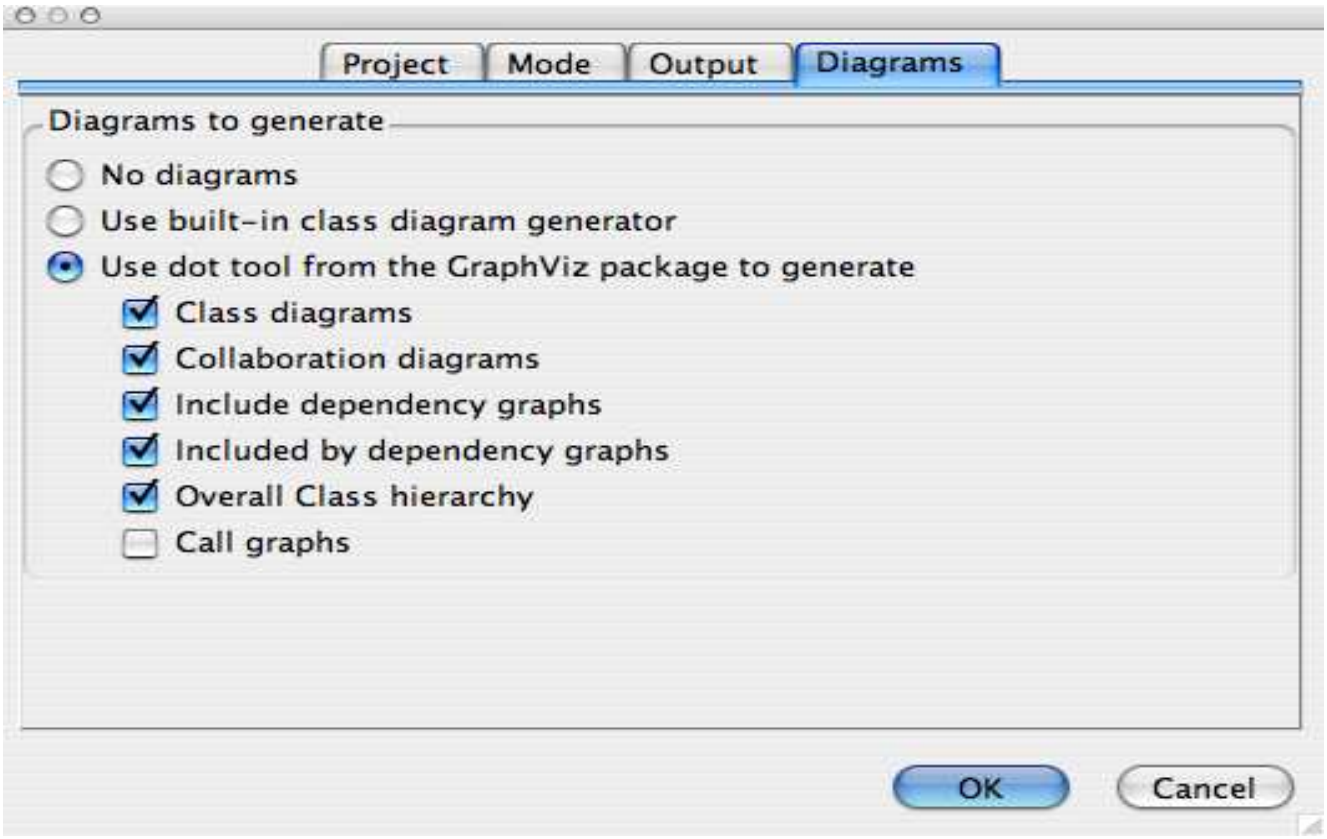
向导对话框：操作模式

模式卡中允许你选择 doxygen 将如何查找你的源码，默认方式只查找被文档化过的内容。
你也能选择 doxygen 如何显示生成的文档，后一个选项不会影响到 doxygen 解析你的源码。



向导对话框：输出生成

你能选择 doxygen 将生成的一种或多种输出格式，HTML 和 LaTex 有一些附带属性。



向导对话框：图表生成

Doxygen 能生成一定数量的图表，使用图表卡你能选择生成那些图表。而且需要 GraphViz 软件包中的 dot 工具来支持大多数图表的生成。（如果你使用的是 MacOSX 的二进制软件包，那么 dot 工具已经包含在其中了）

专家对话框

专家对话框中也包含一定数量的选项卡，每个选项卡对应于配置文件的一节。每个选项卡又包含一定数量的设定行，同时对应于配置小节中每个配置选项。

对应于配置选项类型，以下是对话框中输入部件的种类。

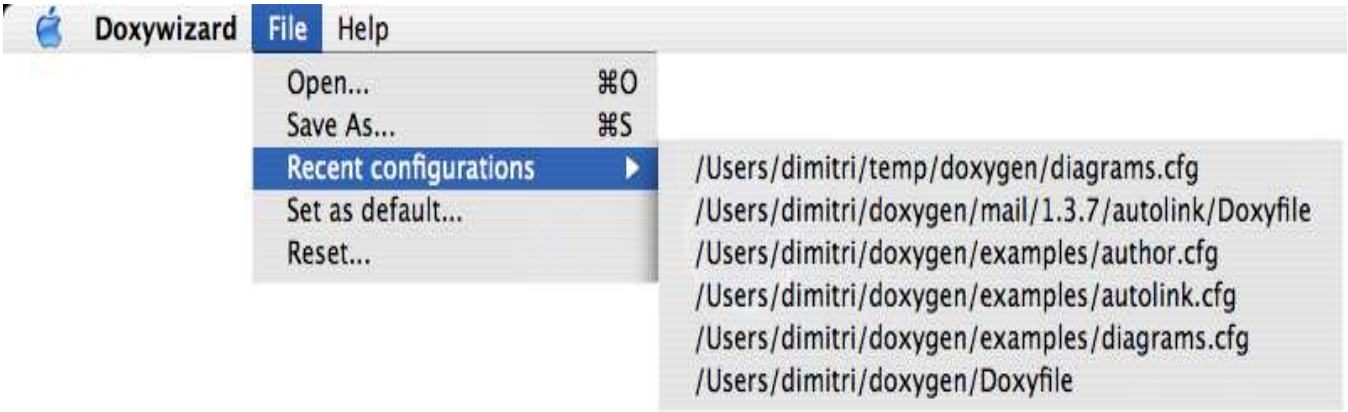
专家对话框中的一些选项

- 每个复选框中的布尔值（等同于配置文件中的 YES 和 NO）
- 组合框用于选择所有条目中的一个固定值。（如 OUTPUT_LANGUAGE）
- 微调按钮用于选择一定范围内的整型值。
- 文本编辑行用于字串类型的选项。
- 字符串列表，使用“+”按钮将文本编辑行中的字串加入到列表中；使用“-”按钮将列表的某行字串删除掉；使用“*”按钮可将文本编辑行中的字串，替换掉当前列表所选中的那行字串。
- 使用特殊的按钮可调用一个文件选择对话框。

在对话框的右下角点击 Help 按钮，将光标放置到选项之上，此时一个冒泡提示将出现，它能提供该选项的附带信息。

菜单选项

图形前端包含有一个菜单，和若干有用的项。



文件菜单

Open..

等同于主窗口中的 Load 按钮，允许从磁盘中打开一个配置文件。

Save as..
等同于主窗口中的 Save 按钮，能将当前的配置设定保存到磁盘中。

Recent configurations
允许快速载入一个最新的已保存的配置文件。

Set as default..
保存当前的配置设定为下一次启动后的默认值，你需要确认此更改方可激活。

Reset..
恢复默认值到初始状态，你需要确认此更改方可激活。

Installdox　用　法

Installbox 是一个 doxygen 生成的 perl 脚本，用于标记文件（查阅“**外部参考选项**”小节中的 TAGFILES），此脚本将被放置到生成的 HTML 文件所在的目录下。

主要的用途是在安装时为每个标记文件设置外部文档的位置。

在命令行中调用 installbox 并附带-h 选项，将会获得该脚本用法的一个简明描述。

以下是一些有效的附带选项：

-l <tagfile>@<location>
每个标记文件中包含了 HTML 文件集中被文档化后的文件，类和成员的信息。用户能将这些 HTML 文件安装到磁盘或 web 页面的任何地方。另外 installbox 需要为每一个标记文件<tagfile>设定文档的位置，使之能被 doxygen 使用。位置<location>可以是一个绝对路径或是一个 URL。

注意，每一个<tagfile>必须是唯一的，且只包含标记文件名，而不包括路径。

-q
当这个选项被指定，installbox 将会生成致命错误的检测输出。

如果需要对这些文件进行扫描和修改，可选择给出一个 HTML 文件的列表，如果在当前目录下所有文件都是 html 文件，那么此文件列表可省略。

为了每个生成类别的浏览，Installbox 脚本将是唯一的，且能够辨析标记文件所使用的部分。The installdox script is unique for each generated class browser in the sense that it `knows` what tag files are used. 如果未附带-l 选项来指定一个标记文件，或者是给出的标记文件无效，installbox 将生成一个错误。

配　　置

格式

配置文件是一个开放的 ASCII 文本文件，且包含一个类似于 Makefile 的文件结构，它的默认名称是 doxyfile，并能被 doxygen 解析。这个文件可以包含若干的制表符和换行符，用于格式化的目的。文件中语句对大小写敏感，能在文件的任何位置放置注释（除了引号之间）。从“#”字符开始的一行内容将被当成注释。

该文件主要包括一个赋值语句列表，每条语句的首部都包括一个 TAG_NAME，并跟随一个“=”符号，以及一个或多个值。如果同一个选项被多次赋值，那么最后的赋值将覆盖掉之前的。对于能附带参数列表的选项，可以使用“+=”操作符来替代“=”，它能将新值添加到参数列表中，而其中的值是一个无空格的连接序列，如果这些值包含一个或多个空格，它必须使用引号（“…”）。在一行的最后一个字符后插入一个反斜杠“\”，可将多个行连接在一起。环境变量使用$(ENV_VARIABLE_NAME)来扩展。

你也能使用@INCLUDE 标记包含其他的配置文件，如下：

```
@INCLUDE = config_file_name
```

Doxygen 将会在当前工作目录下搜索被包含的配置文件，你也能指定一个目录列表，用于在查找当前工作目录之前进行配置文件的搜索。在@INCLUDE 标记之前放置一个@INCLUDE_PATH 标记，如：

```
@INCLUDE_PATH = my_config_dir
```

配置选项可以被分为若干类别，以下是按字母索引排列的有效标记，它之后是按类别区分的标记描述。

（原文此处有一个标记索引列表，因为不准备建立一个文本链接，所以删除。查看 http://www.stack.nl/~dimitri/doxygen/config.html#cfg_abbreviate_brief）

与项目有关的选项

DOXYFILE_ENCODING
指定文件中所有字符的编码格式。在此标记出现之前默认使用 UTF-8。Doxygen 可以使用 libiconv（或在 libc 中创建 iconv）实现编码转换。查阅 http://www.gnu.org/software/libiconv 中列出的编码格式。

PROJECT_NAME
此标记是一个单词（或者使用双引号可包含一个字串），用于标识生成的文档项目，这个名称可用于大多数生成页面的标题或某些地方。

PROJECT_NUMBER
此标记能用于记录一个项目或是修订版本号，能非常容易地将生成的文档进行归类，或是被一些版本控制系统所使用。

OUTPUT_DIRECTORY
指定一个（相对或绝对）的路径，用于生成文档的写入点。如果是一个相对路径，它将会关联到启动 doxygen 的位置。如果留空则表示当前目录将被使用。

CREATE_SUBDIRS
如果此标记设定为 YES，在每种输出格式的输出目录下，doxygen 将创建 4096 个子目录（主输出目录的下 2 层中），并分发生成文件到这些目录中。当提供了一个数量巨大的源码文件集给 doxygen 时，如果将所有的生成文件都放置到同一目录下，这将导致文件系统的性能问题，而此时有效 CREATE_SUBDIRS 选项将非常有益处。

OUTPUT_LANGUAGE
用于指定所有生成文档的语言，doxygen 将使用此信息，生成指定语言的所有常规输出。默认语言是英语，其他支持的语言有：南非荷兰语，阿拉伯语，巴西语，加泰罗尼亚语，汉语，克罗地亚语，捷克语，丹麦语，荷兰语，芬兰语，法语，德语，希腊语，匈牙利语，意大利语，日语，韩语，立陶宛语，挪威语，波斯语，波兰语，葡萄牙语，罗马尼亚语，俄语，塞尔维亚语，斯洛伐克语，斯洛文尼亚语，西班牙语，瑞典语，乌克兰语。

BRIEF_MEMBER_DESC
假如此标记设为 YES（默认值），doxygen 将在文件和类文档（类似于 JavaDoc）中列出的成员之后加入简明的成员描述，若设为 NO 则无效。

REPEAT_BRIEF
如果此标记设为 YES（默认值），doxygen 将会在详细描述之前加入一个成员或函数的简明描述。
注意，若 HIDE_UNDOC_MEMBERS 和 BRIEF_MEMBER_DESC 都设为 NO，简明描述将被完全隐藏。

ABBREVIATE_BRIEF
这个标记采用了一种仿智能的简明描述的缩写工具，用于多个列表中的文本。在列表的每个字串中，如果查找到简明描述的首位字符，会在整个列表处理之后的结果和文本中剔除，可看成为附加说明的文本。否则将维持原状。如果标记之后留空，下面的值将被使用（“\$name”自动替换某些实体的名称）：$name 类，$name 部件，以及$name 文件。If left blank, the following values are used ("\$name" is automatically replaced with the name of the entity): "The $name class" "The $name widget" "The $name file" "is" "provides" "specifies" "contains" "represents" "a" "an" "the".

ALWAYS_DETAILED_SEC
如果 ALWAYS_DETAILED_SEC 和 REPEAT_BRIEF 都设为 YES，在只有一个简明描述的地方 doxygen 将生成一个详细的小节。

INLINE_INHERITED_MEMB

若设为 YES，doxygen 将在类文档中显示类的所有派生成员。并假定这些成员均为普通类成员。基类的构造器，析构器以及赋值操作符将不会显示。

FULL_PATH_NAMES
若设为 YES，doxygen 将在文件列表和头文件中的文件名之前加入完整路径。若设为 NO，将使用能保证文件名唯一性的最短路径。

STRIP_FROM_PATH
若 FULL_PATH_NAMES 设为 YES，STRIP_FROM_PATH 用于去除路径中用户自定义的部分，且只会去除与路径左起匹配的多个指定字串，此标记能够显示文件列表中相关的路径，假如标记后留白此标记无效。

STRIP_FROM_INC_PATH
从类文档内所涉及的路径中，去除用户自定义的部分，它将告知阅读者一个类将包含那些头文件。假如标记后留白，只会使用类定义中的头文件名。否则将会在正常编译时使用-I 参数指定，此标记之后的包含路径。

CASE_SENSE_NAMES
若设为 NO，doxygen 将只生成小写字母的文件名，若设为 YES，允许使用大写字母。如果你的类或文件名之间存在同名但有大小写的差异，以及你的文件系统支持大小写敏感的文件名，那么这个标记是非常有用的。Windows 用户可设为 NO 忽略它。

SHORT_NAMES
若设为 YES，doxygen 将生成最短（但可读）的文件名。当你的文件系统不支持长文件名，如 DOS，Mac，CD-ROM 时，这个标记是非常有用的。

JAVADOC_AUTOBRIEF
若设为 YES，doxygen 将会使用一个 JavaDoc 风格的注释，作为简明描述来表述第一行（直到第一行结束）。若设为 NO(默认值),JavaDoc 风格看上去更象是一个标准的 Qt 注释(这里需要一个@brief 命令来生成一个简明描述)。

QT_AUTOBRIEF
若设为 YES，doxygen 将会使用一个 Qt 风格的注释，作为简明描述来表述第一行（直到第一行结束）。若设为 NO（默认值），Qt 风格看上去更象是一个标准的 Qt 注释（这里需要一个\brief 命令来生成一个简明描述）。

BUILTIN_STL_SUPPORT
如果你使用了 STL 类（如 std:: string，std:: vector 等等），但又不希望包含（在一个标记中）STL 库，那么可将此标记设为 YES，使得 doxygen 能够匹配包含 STL 类的函数声明和定义（例如 func（std:: string）等同于 func（std:: string） {}），这也使得继承和协作图中，涉及到 STL 类的部分更加完整和准确。

CPP_CLI_SUPPORT
假如你使用了微软的 C++/CLI 语言，可设为 YES 使能解析支持。

SIP_SUPPORT
如果你的项目中只包含 SIP 源码可设为 YES，doxygen 解析时会将其看成一般的 C++代码，但是当 protection 关键字没有出现时，会假定所有的类都使用了公有继承而不是私有继承。
** SIP 是一个自动生成 C/C++库的 Python 绑定端的工具，原本是为了 PyQt（Qt GUI 工具包的绑定端）而开发

的，时间始于 1998 年。但它更适合生成任意 C/C++库的绑定端。参考页面：
http://www.riverbankcomputing.co.uk/static/Docs/sip4/introduction.html

IDL_PROPERTY_SUPPORT
微软 IDL 的获取和设定特性，用于表示一个属性获取和设定的方法。设置这个选项为 YES（默认），使得 doxygen 在文档中覆盖掉属性的获取和设定方法，这只能工作在这些方法获取和设定简单类型时。如果不是这种情况，而总是希望显示这些方法，那么将此选项设为 NO。

DISTRIBUTE_GROUP_DOC
如果在文档中使用了成员组，以及此选项设为 YES，为了组中的其他成员，doxygen 将会重复使用用文档中组的第一个成员（如果有），then doxygen will reuse the documentation of the first member in the group (if any) for the other members of the group 组默认情况下，所有的组成员必须被显示文档化。

MULTILINE_CPP_IS_BRIEF
若设为 YES，doxygen 需要处理一个多行的 C++特殊注释块（如，//!或///的注释块），类似于一个简明描述。这是一个默认状态。最新修订后的默认状态则是将上述的特殊注释块看作一个细节描述。如果你接受修改后的默认状态，可将此标记设为 YES，注意，此标记设为 YES，也就意味着 rational rose 的注释无法被完全辨别。

INHERIT_DOCS
若设为 YES(默认值)，那么一个未文档化成员继承于其他文档化成员，它将重新执行文档化。If the INHERIT_DOCS tag is set to YES (the default) then an undocumented member inherits the documentation from any documented member that it re-implements

SEPARATE_MEMBER_PAGES
若设为 YES，doxygen 将为每一个成员建立一个新页，若设为 NO，成员文档将成为文件/类/名字空间文档中的一部分。

TAB_SIZE
设置制表符的大小，doxygen 使用这个数值替换掉代码段中的空格数。

ALIASES
在文档中用于指定一定数量的 alias，等同于命令。一个 alias 有以下的格式，
name=value

这里增加一个例子：
"sideeffect=\par Side Effects:\n"

它允许你放置\sideeffect(或@sideeffect)命令在文档中，这将导致在用户定义的段落标题为“Side Effects:”，你可放置\Side Effects:插入换行符。You can put \n's in the value part of an alias to insert newlines.

OPTIMIZE_OUTPUT_FOR_C
若设为 YES，如果你的项目中只包含 C 源码，doxygen 将为 C 生成更适合的输出，比如，使用一些不同的名称，忽略所有成员的列表，等等。

OPTIMIZE_OUTPUT_JAVA
若设为 YES，如果你的项目中只包含 Java 或 Python 的源码，doxygen 将为这些语言生成更适合的输出，比如出现的名字空间将被看做包，获取认证的范围将会有差异，等等。

OPTIMIZE_FOR_FORTRAN
若设为 YES，如果你的项目中只包含 Fortran 的源码，doxygen 将为它生成更适合的输出。

OPTIMIZE_OUTPUT_VHDL
若设为 YES，如果你的项目中只包含 VHDL 的源码，doxygen 将为它生成更适合的输出。

EXTENSION_MAPPING
Doxygen 使用它能够解析的文件扩展名，来选择解析器， 此标记能为解析器分配一个给定的扩展名，doxygen 有一个内建的映射，但你可以使用此标记覆盖或扩展它。格式为 ext=language，ext 即文件的扩展名，language 则为 doxygen 所支持的解析器中的一个：IDL, Java, Javascript, C#, C, C++, D, PHP, Objective-C, Python, Fortran, VHDL, C, C++。比如使得 doxygen 将.inc 看做 Fortran 文件（默认为 PHP 文件），.f 看做 C 文件（默认为 Fortran 文件），可以使用：inc=Fortran，f=C。

SUBGROUPING
若设为 YES（默认），允许相同类型的类成员组（比如一个公有函数组）被放置到此类型的子分组中（如在 Public Functions 小节下），设为 NO 禁用子分组。另外，对每个类使用\nosubgrouping 命令同样可禁用子分组。

TYPEDEF_HIDES_STRUCT
若设为 YES,文档化的结构,联合或枚举的类型定义,将被看做它们类型的名称。如 typedef struct TypeS {} TypeT，那么文档中就存在一个名为 TypeT 的结构，当无效这个类型定义时，它有可能是文件，名字空间或类的成员，而这个结构将被命令为 TypeS，这对于 C 代码是非常有价值的，可在编码约定中指定所有的混合类型的定义，以及只对此类型定义进行引用，而不需要标记名称。

SYMBOL_CACHE_SIZE
它会决定内部缓存的大小，以此来确定那些符号应保存在内存，而那些应更新到磁盘中。当缓存已满不经常被使用的符号将被写入到磁盘中。对于小中型项目（小于 1000 个输入文件）来说，默认值已经能满足你的要求。对于大型项目来说，缓存过小，有可能导致 doxygen 频繁与磁盘交换符号而花费大量的时间，而造成性能的下降。如果系统中有足够的物理内存，可增加缓冲来保存更多的符号以改善性能。注意,缓存实际大小在一个对数范围内变化，所以增加的缓存大小应该会超过原来的一倍。缓冲大小的计算公式：$2^{(16+SYMBOL\_CACHE\_SIZE)}$。有效的范围是 0~9，默认为 0，相应缓存的大小为 $2^{16} = 65536$。

与创建有关的选项

EXTRACT_ALL
若设为 YES, doxygen 会假定文档中的所有主体将被文档化，即使在未提供文档的情况下。私有类成员和文件静态成员将隐藏，直到 EXTRACT_PRIVATE 和 EXTRACT_STATIC 标记设为 YES。
注意：

当 WARNINGS 设为 YES 时，将无效生成过程中未文档化成员发出的警告。

EXTRACT_PRIVATE
若设为 YES，在文档中将包含所有类私有成员。

EXTRACT_STATIC
若设为 YES，在文档中将包含所有文件的静态成员。

EXTRACT_LOCAL_CLASSES
若设为 YES，在文档中将包含在源文件中定义的类（和结构），若设为 NO，只包含在头文件中定义的类。对于 Java 源码无任何影响。

EXTRACT_ANON_NSPACES
若设为 YES，匿名空间的成员将被提取到文档中，并被称为 anonymous_namespace{file}的名字空间，file 可使用匿名空间包含的文件基类名进行替换。默认情况下，匿名空间将被隐藏。

EXTRACT_LOCAL_METHODS
这个标记只对 Object-C 代码有效，当设为 YES 时，在文档中包含的局部方法将在执行部分被定义，而不会在接口中。若设为 NO（默认），只有在接口中的方法才会被包含到文档。

HIDE_UNDOC_MEMBERS
若设为 YES，doxygen 将隐藏在文档化的类或文件中的所有未文档化的成员，若设为 NO（默认），这些成员将包含在一些预览中，但不会有文档小节生成，如果 EXTRACT_ALL 为 YES，此选项无效。

HIDE_UNDOC_CLASSES
若设为 YES，doxygen 将隐藏所有未文档化的类，若设为 NO（默认），这些类将被包含在一些预览中，但不会有文档小节生成，如果 EXTRACT_ALL 为 YES，此选项无效。

HIDE_FRIEND_COMPOUNDS
若设为 YES，doxygen 将隐藏所有友元（类|结构|联合）声明，若设为 NO（默认），这些声明将包含在文档中。

HIDE_IN_BODY_DOCS
若设为 YES，doxygen 将隐藏函数主体中的一些文档块，若设为 NO（默认），这些块将出现在函数的细节文档块中。

INTERNAL_DOCS
用于确认在一个\internal 命令之后能否包含文档。若设为 NO（默认），那么文档将被移除，如设为 YES，将包含内部文档。

HIDE_SCOPE_NAMES
若设为 NO（默认），doxygen 将显示文档中类成员和名字空间的作用域，若设为 YES，此作用域将被隐藏。

SHOW_INCLUDE_FILES
若设为 YES（默认），doxygen 将在文件的文档中放置它所包含文件的列表。

FORCE_LOCAL_INCLUDES
若设为 YES，doxygen 将会在文档中使用双引号列出所包含的文件，而不是尖括号。

INLINE_INFO
若设为 YES（默认），将在文档中插入 inline 标记来指示 inline 成员。

SORT_MEMBER_DOCS
若设为 YES（默认），doxygen 将按英文字母的顺序对文件和类的成员名字进行文档排序，若设为 NO，成员将按声明的顺序出现。

SORT_BRIEF_DOCS
若设为 YES，doxygen 会按英文字母的顺序将文件，名字空间和类的成员名字，作为它们简明描述的排序依据，若设为 NO（默认），成员将按声明的顺序出现。

SORT_GROUP_NAMES
若设为 YES，doxygen 会按英文字母的顺序对组名的层级进行排序。若设为 NO（默认），组名将按定义的顺序出现。

SORT_BY_SCOPE_NAME
若设为 YES，将由类的全称（即包含名字空间）对类列表进行排序。若设为 NO（默认），类列表排序只能由不包含名字空间的类名决定。
注意：
如果 HIDE_SCOPE_NAMES 设为 YES，这个选项不太有价值。
此选项只适用于类列表，且不是按字母排序的列表。

SORT_MEMBERS_CTORS_1ST
若设为 YES，doxygen 将对类成员的（简明和细节描述）文档进行排序，且构造器和析构器将位于列表首部。若设为 NO（默认），构造器将出现在 SORT_MEMBER_DOCS，SORT_BRIEF_DOCS 所设定的各自定义的位置。
注意：
若 SORT_BRIEF_DOCS 设为 NO，此选项将忽略成员简明描述的排序。
若 SORT_MEMBER_DOCS 设为 NO，此选项将忽略成员细节描述的排序。

GENERATE_DEPRECATEDLIST
用于有效（YES）或失效（NO）废弃列表（主要针对类和类方法），通过放置\deprecated 命令来创建废弃列表。

GENERATE_TESTLIST
用于有效（YES）或失效（NO）测试列表，通过放置\test 命令来创建测试列表。

GENERATE_BUGLIST
用于有效（YES）或失效（NO）bug 列表，通过放置\bug 命令来创建 bug 列表。

ENABLED_SECTIONS
用于有效某些条件化文档小节，可使用标记，\if <小节标号> … \endif 和\cond <小节标号> … \endcond 的块结构。

MAX_INITIALIZER_LINES
用于确定一个变量或定义初始化时能使用的最大文本行数，如果初始化时的行数超过最大值，那么它将被隐藏。将最大值设为 0 将完全隐藏初始化的文本。可在文档中使用\showinitializer 和\hideinitializer 命令，来控制个别变量和定义的初始化文本。

SHOW_USED_FILES
若设为 NO，在类和结构文档的底部无效文件列表的生成。若设为 YES，列表将会提取那些已被用于生成文档的文件。

SHOW_DIRECTORIES
如果你项目中源码分布在多个目录下，可设定此选项为 YES，在文档中显示出源码目录的层级。

SHOW_FILES
若设为 NO，将无效文件页面的生成，并将文件从快速索引和文件树浏览（如果指定）中移除。默认设定为 YES。

SHOW_NAMESPACES
若设定为 NO，将无效名字空间页面的生成，并将名字空间从快速索引和文件树浏览（如果指定）中移除。默认设定为 YES。

与警告和处理消息相关的选项

QUIET
用于打开或关闭 doxygen 生成输出时的消息，若设定 YES 表明关闭消息，若留白则设定为 NO。

WARNINGS
用于打开或关闭 doxygen 生成错误记录时的警告消息，若设定 YES 表明打开警告消息，若留白则设定为 NO。
技巧：当写文档时应打开警告消息。

WARN_IF_UNDOCUMENTED
若设为 YES，doxygen 将为未文档化的成员生成警告消息，如果 EXTRACT_ALL 设为 YES，那么该标记将自动失效。

WARN_IF_DOC_ERROR
若设为 YES，doxygen 将为文档中潜在的错误生成警告消息，例如在一个文档化函数中并没有文档化一些参数，或文档化的参数并不存在，又或者使用了错误的标记命令。

WARN_NO_PARAMDOC
用于获得文档化函数所生成警告消息，但不能兼顾到它的参数和返回值。若设为 NO（默认）doxygen 只能生成错误或参数文档不完整的警告，而不能监控文档的缺失。

WARN_FORMAT
确定 doxygen 所生成警告消息的格式，消息字串会包含$file, $line, $text 标记，并且这些标记将被发出警告

的文件名，行号，以及相应的文本所替换。

WARN_LOGFILE
用于指定一个警告和错误消息写入的日志文件，如果留空表示将消息写入 stderr。

与输入相关的选项

INPUT
用于指定将被文档化的源文件和目录，你可以输入 myfile.cpp 文件名或/usr/src/myproject 目录名，多个文件名或目录名可用空格进行分离。

注意：如果此标记留空，那么将搜索当前目录。

INPUT_ENCODING
用于指定 doxygen 解析源文件所使用字符编码，doxygen 内部使用 UTF-8 编码，同样也是默认的输入编码。Doxygen 可使用 libiconv（或者在 libc 中创建 iconv）进行编码转换。查看 http://www.gnu.org/software/libiconv/ 中编码列表。

FILE_PATTERNS
如果 INPUT 包含有目录，可用此选项指定一个或多个通配符（如*.cpp 和*.h）在目录过滤源文件。假如选项后留空，以下的扩展名将用于过滤：.c *.cc *.cxx *.cpp *.c++ *.java *.ii *.ixx *.ipp *.i++ *.inl *.h *.hh *.hxx *.hpp .h++ *.idl *.odl *.cs *.php *.php3 *.inc *.m *.mm 。

FILE_VERSION_FILTER
用于指定一个程序或脚本，使得 doxygen 能调用每个文件以获得当前版本（一般通过 CVS 版本控制系统获取），且 doxygen 将会执行（使用 popen（））command input-file 命令来调用该程序，其中 command 为 FILE_VERSION_FILTER 后跟的设定名称，input-file 则为 doxygen 所支持的输入文件名。无论该程序向标准输出写入任何内容，都将被用做文件版本。

Unix 下使用一个 shell 脚本过滤器的例子:
FILE_VERSION_FILTER = "/bin/sh versionfilter.sh"

CVS 的脚本例子：
```
#!/bin/sh
cvs status $1 | sed -n 's/^[ \]*Working revision:[ \t]*\([0-9][0-9\.]*\).*/\1/p'
```

SVN 的脚本例子：
```
#!/bin/sh
svn stat -v $1 | sed -n 's/^[ A-Z?\*|!]\{1,15\}/r/;s/ \{1,15\}/\/r/;s/ .*//p'
```

ClearCase 过滤器的例子：

FILE_VERSION_INFO = "cleartool desc -fmt \%Vn"

LAYOUT_FILE
用于指定一个能被 doxygen 解析的布局文件。在一种独立的输出格式中，此布局文件可以控制所生成的输出文件的全局结构。创建默认的布局文件可运行 doxygen 附带-l 选项，并且可选择在选项之后是否指定一个文件名，如果文件名被忽略将使用 DoxygenLayout.xml，注意，如果你运行 doxygen 的目录中已经包含了一个名为 DoxygenLayout.xml 的文件时，倘若 LAYOUT_FILE 之后留空 doxygen 将自动解析该文件。

RECURSIVE
用于是否在子目录中搜索可用的输入文件，如果选项之后留空则设为 NO。

EXCLUDE
用于指定那些文件和目录将从 INPUT 选项中将被移除。这种方式使你很容易得从 INPUT 选项设定根目录树中，移除一个子目录。

EXCLUDE_SYMLINKS
用于选择是否从输出中移除文件或者目录的符号链接（一个 Unix 文件系统的特性）。

EXCLUDE_PATTERNS
在 INPUT 选项设定的目录中，使用该选项指定的一个或多个通配符来移除匹配的文件。
注意，通配符将对文件的绝对路径进行匹配，如移除所有的测试目录，可使用*/test/*的样式。

EXAMPLE_PATH
指定一个或多个包含例子代码段的文件或包含例子文件的目录。（在\include 小节中查看\include 命令）

EXAMPLE_RECURSIVE
若设为 YES，将在子目录中搜索输入例子文件，可使用\include 或\dontinclude 命令，它与 RECURSIVE 无关，如果选项之后留空则设为 NO。

EXAMPLE_PATTERNS
若 EXAMPLE_PATH 选项之后含有目录，可用此标记指定一个或多个通配符（如*.cpp 和*.h），从上述目录中过滤掉匹配的文件，如果选项后留空则设为 NO。

IMAGE_PATH
用于在文档中指定一个或多个包含图片的文件或是包含图片文件的目录。（查看\image 命令）

INPUT_FILTER
指定一个程序，使得 doxygen 对每一个输入文件进行过滤，doxygen 调用过滤程序可以执行（使用 popen()）这个命令：
<filter> <input-file>

<filter>则为 INPUT_FILTER 的设定值，<input-file>则为输入文件名，doxygen 会将过滤程序的输出写入到标准输出中。

FILTER_PATTERNS
用于在每种文件样式的基础上指定过滤规则。Doxygen 将会比较文件名和每一种样式，并检查是否与过滤规则匹配。这些过滤规则来自一个列表：pattern=filter（如*.cpp=my_cpp_filter），查看 INPUT_FILTER 进一步获得如何使用过滤器的信息，假设选项后留空，INPUT_FILTER 将用于全部文件。

FILTER_SOURCE_FILES
若设为 YES，输入过滤器（可使用 INPUT_FILTER 设定）既可过滤输入文件，也可用于生成源文件的浏览（当 SOURCE_BROWSER 设为 YES 时）。

与源码浏览相关的选项

SOURCE_BROWSER
若设为 YES，将生成一个源码文件的列表。文档主体部分将对这些源码进行交叉引用。
注意，在生成输出的过程中与所有源码的关联有可能失效，因此要确认 VERBATIM_HEADERS 是否设为 NO。

INLINE_SOURCES
若设为 YES，在文档中将直接包含函数，类，枚举的代码段。

STRIP_CODE_COMMENTS
若为 YES（默认），告知 doxygen 隐藏一些来自源码段中特殊的注释块，正常情况下 C 和 C++的注释始终可见。

REFERENCED_BY_RELATION
若为 YES，所有的文档化函数都将会列出其引用过的函数。

REFERENCES_RELATION
若为 YES，所有的文档化实体都将会列出其调用或使用过的函数。

REFERENCES_LINK_SOURCE
若为 YES（默认），且 SOURCE_BROWSER 也为 YES，那么 REFERENCES_RELATION，REFERENCED_BY_RELATION 所生成列表中函数的超链接将指向源码，否则将指向该函数的文档。

VERBATIM_HEADERS
若为 YES（默认），doxygen 将为类所指定的头文件，生成一个相同的副本，doxygen will generate a verbatim copy of the header file for each class for which an include is specified 若为 NO 则无效。
也可查看\class 小节。

USE_HTAGS
若为 YES，对于源码的引用，将指向由 htags 工具而非 doxygen 内建的源码浏览器生成的 HTML。Htags 工具是 GNU 全局源码标签系统 GNU's global source tagging system 的一部分（参看 http://www.gnu.org/software/global/global.html）。按以下步骤使用 htags 工具：
1. 安装最新的版本（如 4.8.6 或更新的）

2. 有效 SOURCE_BROWSER，USE_HTAGS
3. 确认 INPUT 指向的是源码树的根目录
4. 正常运行 doxygen

Doxygen 将调用 htags（依次调用 gtags），这些工具只有在命令行下才有效（并存在于系统的检索路径中）。得到的结果，可替代 doxygen 所生成的源码浏览器，指向源码的链接将会指向 htags 的输出。

与字母索引相关的选项

ALPHABETICAL_INDEX
若为 YES，将会生成一个所有合成部分的字母索引。如果项目中包含一些类，结构，联合和接口时，应有效此标记。

COLS_IN_ALPHA_INDEX
如果 ALPHABETICAL_INDEX 有效，那么此标记可用于指定每列的间距（间距的范围是 1~20）。

IGNORE_PREFIX
项目中所有的类都是由一个公共的前缀开始，那么所有的类也将被放置到字母索引中相同的首部之下。此标记可指定一个特殊的前缀（或是一个前缀列表），当生成索引首部时却可以忽略掉。

与 HTML 相关的选项

GENERATE_HTML
若为 YES（默认），doxygen 将生成 HTML 输出。

HTML_OUTPUT
指定 HTML 文档所存放的位置。如果此标记后为一相对路径，则 OUTPUT_DIRECTORY 的值将被放置到它之前。如果标记后留空，则将 html 作为默认路径。

HTML_FILE_EXTENSION
指定生成的 HTML 页面文件的扩展名（例如：.htm，.php，.asp），如果标记后留空，doxygen 将生成 .html 文件。

HTML_HEADER
指定生成的 HTML 页面中用户自定义的页头。为获得有效 HTML 页头至少要包含一个<HTML>和一个<BODY>标记，但更好的办法是，包含 doxygen 生成的样式表。最小化的例子:

```
<HTML>
    <HEAD>
      <TITLE>My title</TITLE>
```

```
    <LINK HREF="doxygen.css" REL="stylesheet" TYPE="text/css">
  </HEAD>
  <BODY BGCOLOR="#FFFFFF">
```

如果选项后留空，doxygen 将生成一个标准页头。

以下是包含在页头和页脚中具有特定意义的标识：
$title　替换页面的标题
$datetime　替换当前的日期和时间
$date　替换当前的日期
$year　替换当前的年份
$doxygenversion　替换 doxygen 的版本
$projectname　替换项目的名称（查看 PROJECT_NAME）
$projectnumber　替换项目的编号（查看 PROJECT_NUMBER）
$relpath$　如果 CREATE_SUBDIRS 有效，$relpath$命令将生成一个关联到 HTML 输出目录的相对路径，如使用$relpath$doxygen.css,来引用一个标准的样式表。

查看“doxygen 用法”小节获得如何使用 doxygen 生成默认页头的信息。

HTML_FOOTER
指定生成的 HTML 页面中用户自定义的页脚，为获得有效 HTML 页脚至少要包含一个</BODY>和一个</HTML>标记，一个最小化的例子：

```
  </BODY>
</HTML>
```

如果选项后留空，doxygen 将生成一个标准页脚。

在页脚中具有特定意义的命令有:$title, $datetime, $date, $doxygenversion, $projectname, $projectnumber Doxygen 将分别用页标题，当前日期和时间，当前日期，doxygen 的版本号，项目名（查看 PROJECT_NAME），项目编号（查看 PROJECT_NUMBER）来替换这些命令。

查看“doxygen 用法”小节获得如何使用 doxygen 生成默认页脚的信息。

HTML_STYLESHEET
为 HTML 页面指定一个用户自定义的样式表，它可用于 HTML 输出观感的微调。如果标记后留空，doxygen 将生成默认的样式表。

查看“doxygen 用法”小节获得如何使用 doxygen 生成默认样式表的信息。

HTML_TIMESTAMP
若为 YES，生成的 HTML 页面的页脚中将包含页面生成的日期和时间。设为 NO，当比较多次运行后生成的输出时可提供帮助。

HTML_ALIGN_MEMBERS

若为 YES，HTML 的类，文件或名字空间的成员使用表格对齐，若为 NO 将使用一个无序列表。

注意：此标记设为 NO 在未来将被废弃，目前我只关注于支持和测试那些已对齐的文本描述。

HTML_DYNAMIC_SECTIONS
若为 YES，在生成的 HTML 文档装载之后，它将会包含可隐藏和显示的动态部分。这一特性需要浏览器支持 JavaScript 和 DHTML（例如 Mozilla 1.0+, Firefox Netscape 6.0+, Internet explorer 5.0+, Konqueror, Safari）。

GENERATE_DOCSET
若为 YES，将为运行于 OSX 10.5 (Leopard)上的 Apple Xcode 3（http://developer.apple.com/tools/xcode/）集成开发环境生成可用的索引文件，创建一个文档集，doxygen 将在 HTML 输出目录中生成一个 Makefile，并在目录中运行 make 创建 docset，然后运行 make intall 将 docset 安装到 ~/Library/Developer/Shared/Documentation/DocSets 下，使得 Xcode 在启动时能够找到它。查看 http://developer.apple.com/tools/creatingdocsetswithdoxygen.html 获得更多的信息。

DOCSET_FEEDNAME
若为 YES，确认 feed 的名称，一个文档 feed 支持将多个来自一个提供者的文档组合成一个整体。（如同一间公司或一组产品套件）A documentation feed provides an umbrella under which multiple documentation sets from a single provider (such as a company or product suite) can be grouped

DOCSET_BUNDLE_ID
若 GENERATE_DOCSET 为 YES，此标记将指定一个字串，用于文档集被捆绑后的唯一标识。this tag specifies a string that should uniquely identify the documentation set bundle 而它是一个将域名颠倒的字串，比如 com.mycompany.MyDocSet。doxygen 将附加.docset 到字串后。

DOCSET_PUBLISHER_ID
当 GENERATE_PUBLISHER_ID 指定一个字串，用于文档提供者的唯一标识。而它是一个将域名颠倒的字串，比如 com.mycompany.MyDocSet.documentation

DOCSET_PUBLISHER_NAME
GENERATE_PUBLISHER_NAME 将标识文档的提供者。

GENERATE_HTMLHELP
若为 YES，doxygen 将生成三个附带的 HTML 索引文件：index.hhp, index.hhc, index.hhk。index.hhp 是一个可被微软 HTML Help Workshop 读取的项目文件。

HTML Help Workshop 包含一个编译器，它可转换所有 doxygen 生成的 HTML 输出文件，成为一个单一的已编译的 HTML 文件（.chm）。目前使用的已编译 HTML 文件被当作 windwos98 帮助文件的格式，在将来的 windows 所有平台上，将会替代老的 windows 帮助文件格式（.hlp）。已压缩的 HTML 文件包含一个索引，一个目录，并且你可以在文档中检索关键字。HTML Help Workshop 也包括了一个已压缩 HTML 文件的查看器。

CHM_FILE
若 GENERATE_HTMLHELP 为 YES，此选项用于指定.chm 输出文件的名称，如果该文件无法写入 html 输出目录，可

在文件前加入一个路径。

HHC_LOCATION
若 GENERATE_HTMLHELP 为 YES，该标记用于指定 HTML 帮助文件编译器（hhc.exe）的本地位置（包含文件名的绝对路径），如果标记后不为空，在生成 index.hhp 时 doxygen 将尝试运行 HTML 帮助文件编译器。

GENERATE_CHI
若 GENERATE_HTMLHELP 为 YES，该标记为 YES，将生成一个单独的.chi 索引文件，若为 NO，.chi 文件将被包含在一个.chm 文件中。

BINARY_TOC
若 GENERATE_HTMLHELP 为 YES，该标记为 YES，将生成一个二进制的目录，若为 NO，将在.chm 文件中生成一个正常的目录。

TOC_EXPAND
若为 YES，可在 HTML 帮助文档的目录和树型查看器中增加额外组成员的条目。

GENERATE_QHP
若为 YES，并且 QHP_NAMESPACE，QHP_VIRTUAL_FOLDER 都已设定，将生成一个附加的索引文件，可使得 Qt qhelpgenerator 从已生成的 HTML 文档中生成一个 Qt 压缩帮助文件（.qch）。

QHP_NAMESPACE
当生成 Qt 帮助文件输出时，指定名字空间，为获得更多信息，请查看
http://doc.trolltech.com/qthelpproject.html#namespace

QHP_VIRTUAL_FOLDER
当生成 Qt 帮助文件输出时，指定名字空间，为获得更多信息，请查看
http://doc.trolltech.com/qthelpproject.html#virtual-folders

QHP_CUST_FILTER_NAME
指定增加的一个自定义过滤器的名称，为获得更多信息，请查看
http://doc.trolltech.com/qthelpproject.html#custom-filters

QHP_CUST_FILTER_ATTRS
指定增加的自定义过滤器的属性列表，为获得更多信息，请查看
http://doc.trolltech.com/qthelpproject.html#custom-filters

QHP_SECT_FILTER_ATTRS
指定与项目过滤器匹配的属性列表，为获得更多信息，请查看
http://doc.trolltech.com/qthelpproject.html#filter-attributes

QHG_LOCATION
若 GENERATE_QHP 为 YES,此标记用于指定 Qt qhelpgenerator 的本地位置,如果标记后不为空,doxygen 在生成.qhp 文件时将尝试运行 qhelpgenerator。

GENERATE_ECLIPSEHELP
若为 YES，将生成附加的索引文件，并与 HTML 文件一同打包成一个 Eclipse 的帮助插件。

在 Eclipse 的帮助菜单下可安装和有效此插件，目录中包含的 HTML 和 XML 文件需要复制到 Eclipse 的插件目录中，且插件目录中的目录名要与 ECLIPSE_DOC_ID 相同。

复制文件之后将要重启 Eclipse 才能使帮助插件出现在开发环境中。

ECLIPSE_DOC_ID
指定 Eclipse 帮助插件的唯一标识。当安装的插件中含有 HTML 和 XML 文件的目录名称将与此标识相同，每一个文档集都必须有它自己的标识。

SEARCHENGINE
若此标记有效，doxygen 将为 HTML 输出生成一个搜索工具。底层的搜索引擎会使用 JavaScript 和 DHTML，且能工作在一些主流的浏览器中。注意，当使用 HTML 帮助文件(GENERATE_HTMLHELP)，Qt 帮助文件（GENERATE_QHP），或者 docset（GENERATE_DOCSET）时，这些文件都已经附带有一个搜索功能，所以通常情况下此标记将无效。对于大型项目，基于 JavaScript 搜索引擎的检索速度将会很慢，可以使能 SERVER_BASED_SEARCH，它可以提供一种更好的解决方法。

可以使用键盘进行检索，使用功能键 +S 可直接调用搜索工具（具体使用那一个功能键，与操作系统和浏览器有关，通常是 CTRL，或 ALT，或者二者皆可）。在搜索工具中使用光标下键，可跳至搜索结果窗口，并使用光标键查看搜索结果。按 Enter 键可选中一栏，使用 Esc 可取消搜索。当光标处于搜索工具内并按下 Shift + 光标下键，可选择过滤器参数，使用光标键选定一个过滤器之后，可按下 Enter 和 Esc 激活或者取消过滤器选项。

SERVER_BASED_SEARCH
若此标记有效，搜索引擎将使用基于 PHP 的 web 服务器模式来替代基于 JavaScript 的 web 客户端模式，doxygen 将在 web 服务器上放置已生成的 PHP 搜索脚本和索引文件。服务器模式的优点在于，对于大型项目的良好表现，以及文本检索。而缺点是安装比较有难度，并且还不具备智能搜索的能力。

DISABLE_INDEX
如果你希望完全掌控已生成 HTML 的页面布局，就必须用自定义的设置替换原有的索引。此标记可用于打开或取消每个页面顶端的精简索引。若为 NO（默认）将使能该索引，若为 YES 则取消。

ENUM_VALUES_PER_LINE
用于设定 doxygen 在一行中可排列多少个枚举值（范围是 1~20 个）。

GENERATE_TREEVIEW
在显示生成的层次信息时，是否采用树型结构。若为 YES，一侧的面板将包含一个树型结构（如同一个生成的 HTML 帮助文件），此类特性需要浏览器支持 JavaScript，DHTML，CSS，frames（如一些主流的浏览器），windows 用户使用 HTML 帮助文件可能是更好的选择。

通过自定义样式表（查看 HTML_STYLESHEET）能进一步微调索引的观感， doxygen 生成的默认样式表中有一个例子，它将显示如何在索引树的根节点放置一副图片来替换 PROJECT_NAME

USE_INLINE_TREES
有效该标记，doxygen 将使用树型视图来生成组，目录，类的层次页面，用以替代一个顺序列表。

TREEVIEW_WIDTH
如果树型视图有效（查看 GENERATE_TREEVIEW），该标记用于设定在树型索引显示的框体的初始宽度（像素值）。

EXT_LINKS_IN_WINDOW
若为 YES，doxygen 将通过导入标记文件，开放指向一个独立窗口的外部符号链接。

FORMULA_FONTSIZE
用于改变 HTML 文档中包含的如同图片一样的 Latex 公式的字体大小。默认为 10，你希望在 doxygen 运行之后成功更改字体的大小，你必须从 HTML 输出目录中手动删除一些 form_*.png 图片，并确认它们被重新生成。

FORMULA_TRANSPARENT
为了显示公式，用于确认是否将图片生成为背景透明的 PNG 文件，透明的 PNG 有可能不被 IE6.0 所支持，但其他所有的主流浏览器都支持。注意，当你更改了此选项，你必须在更改生效之前，在 HTML 输出目录中删除一些 form_*.png 文件。

与 LaTex 相关的选项

GENERATE_LATEX
若为 YES（默认），doxygen 将生成 LaTex 输出。

LATEX_OUTPUT
指定 LaTex 文档放置的位置，如果键入的是一个相对路径，那么 OUTPUT_DIRECTORY 设定值将放置它之前。若标记后留空，将使用默认路径 latex。

LATEX_CMD_NAME
指定将被调用的 LaTex 命令字，若标记后留空，将使用默认命令字 latex。注意，当有效 USE_PDFLATEX 选项只能在 HTML 输出中用于生成位图公式，无法在 Makefile 中使用并写入输出目录。

MAKEINDEX_CMD_NAME
指定生成 latex 索引的命令字，如果标记后留空，将使用默认命令字 makeindex。

COMPACT_LATEX
若为 YES，doxygen 将生成压缩的 latex 文档，这对于小型项目非常有用，以及在通常情况下保存一些树也是非常有帮助的。

PAPER_TYPE
用于指定打印机所使用页面类型，以下是一些标准值：

- a4 (210 x 297 毫米).
- a4wide (大小与 a4 相同, 但包含 a4wide 包).
- letter (8.5 x 11 英寸).
- legal (8.5 x 14 英寸).
- executive (7.25 x 10.5 英寸)

若标记后留空将使用 a4wide。

EXTRA_PACKAGES
指定在 latex 输出中包含一个或多个 latex 包的名称，为获得 times 字体你可以设定
EXTRA_PACKAGES = times
若标记后留空将不包含额外的包。

LATEX_HEADER
指定在生成 latex 文档中包含一个个人风格的 latex 头，在第一章开始之前此头可包含任何内容。

如果标记后留空，doxygen 将生成一个标准头，查看“doxygen 用法”小节，获得如何让 doxygen 写入一个默认头到单独文件的信息。

注意：
如果你知道自己在做什么的话，可以只使用一个用户自定义的头。

之后是头中一些有特殊意义的命令：$title, $datetime, $date, $doxygenversion, $projectname, $projectnumber。Doxygen 可分别使用页标题，当前日期和时间，当前日期，doxygen 版本号，项目名（查看 PROJECT_NAME），项目编号（查看 PROJECT_NUMBER）来替换这些命令。

PDF_HYPERLINKS
若为 YES，latex 将转换生成的文档为 PDF 文件（使用 ps2pdf 或者 pdflatex），PDF 文件将包含链接（如同 HTML 输出）而非页面引用。这使得用 PDF 浏览器作为在线浏览器变得更适合。

USE_PDFLATEX
若为 YES，doxygen 将使用 pdflatex 直接从 latex 文件生成 PDF 文件。

LATEX_BATCHMODE
若为 YES，doxygen 将已生成的 latex 文件中添加\batchmode 命令，它将指示 latex 在出现错误时继续运行，而不是询问用户以获得帮助。此选项也可用于 HTML 中生成公式。

LATEX_HIDE_INDICES
若为 YES，doxygen 将不会在输出中包含索引部分（比如文件索引，复合索引等等）。

与 RTF 相关的选项

GENERATE_RTF
若为 YES，doxygen 将生成 RTF 输出。此 RTF 只针对 Word97 进行了优化，但在其他阅读和编辑器中未必表现良好。

RTF_OUTPUT
指定 RTF 文档放置的位置，如果它是一个相对路径，那么 OUTPUT_DIRECTORY 设定值将放置它之前。若标记后留空，将使用默认路径 rtf。

COMPACT_RTF
若为 YES，doxygen 将生成压缩的 RTF 文档，这对于小型项目非常有用，以及在通常情况下保存一些树也是非常有帮助的。

RTF_HYPERLINKS
若为 YES，生成的 RTF 将包括超链接的区域，RTF 文件将会包含链接（如同 HTML 输出）而非页面引用。这使得用 Word 或是其他一些与 Word 兼容并支持超链接的阅读器，作为在线浏览器变得更适合。

注意：WordPad 和其他兼容的工具都不支持超链接。

RTF_STYLESHEET_FILE
从文件中装载样式表定义，风格与 doxygen 配置文件类似，如同一个配置序列。你只需要提供代换值，缺省的定义将会使用它们的默认值。

查看“doxygen 用法”小节获得如何在 doxygen 正常使用时生成默认样式表的信息。

RTF_EXTENSIONS_FILE
将在 RTF 文档的生成过程中使用该选项的设定值。风格与 doxygen 配置文件类似，一个模板扩展文件可使用 doxygen -e rtf extensionFile 来生成。

与 Man 页面相关的选项

GENERATE_MAN
若为 YES（默认），doxygen 将为类和文件生成 man 页面。

MAN_OUTPUT
指定 man 页面放置的位置。如果它是一个相对路径，那么 OUTPUT_DIRECTORY 设定值将放置它之前。若标记后留空，将使用默认路径 man。使用此选项可在目录中创建一个 man3 目录。

MAN_EXTENSION
确认添加到已生成 man 页面的扩展部分（默认为程序的第三小节）。

MAN_LINKS

若为 YES，doxygen 会生成 man 输出，它将为 man 页面中的文档主体，生成一个附带的 man 文件，这些附带文件来自实际的 man 页面，但没有它们 man 将无法查找到正确的页面。默认为 NO。

与 XML 相关的选项

GENERATE_XML
若为 YES，doxygen 将生成一个 XML 文件，用于表述所有文档中包含的代码结构。

XML_OUTPUT
指定 XML 页面放置的位置。如果它是一个相对路径，那么 OUTPUT_DIRECTORY 设定值将放置它之前。若标记后留空，将使用默认路径 xml。

XML_SCHEMA
用于指定一个 XML 例表 schema，XML 解析器可用其检查 XML 文件的语法。

XML_DTD
用于指定一个 XML DTD，XML 解析器可用其检查 XML 文件的语法。

XML_PROGRAMLISTING
若为 YES，doxygen 会将程序列表（包含高亮语法和交叉引用的信息）转存到 XML 的输出。注意，此标记有效将会显著增加 XML 输出的大小。

与 AUTOGEN_DEF 相关的选项

GENERATE_AUTOGEN_DEF
若为 YES，doxygen 将会生成一个 AutoGen 的定义（查看 http://autogen.sf.net）文件，用于表述所有文档中包含的代码结构。注意，此特性一直在试验中，目前还不完善。

与 PERLMOD 相关的选项

GENERATE_PERLMOD
若为 YES，doxygen 将生成一个 Perl 模块，用于表述所有文档中包含的代码结构。注意，此特性一直在试验中，目前还不完善。

PERLMOD_LATEX

若为 YES，doxygen 将生成必要的 Makefile 规则，以使 Perl 脚本和 latex 代码可从 Perl 模块的输出中生成 PDF 或 DVI 文件。

PERLMOD_PRETTY
若为 YES，Perl 模块的输出将具有良好的格式，以方便使用者阅读。对于你理解它的操作过程也是非常有价值的。另类情况下，若为 NO，Perl 模块的输出规模将非常小，但对 Perl 来说是可解析的。

PERLMOD_MAKEVAR_PREFIX
在已生成 doxyrules.make 文件中的 make 变量名，将被加入到此选项所设定的字串之前。The names of the make variables in the generated doxyrules.make file are prefixed with the string contained in PERLMOD_MAKEVAR_PREFIX.这对于在同一 Makefile 中包含有若干个不同的 doxyrules.make，且相互存在不能覆盖的变量名时，这个选项将非常有用。

与预处理有关的选项

ENABLE_PREPROCESSING
若为 YES（默认），doxygen 将在源文件和头文件中，判别所有的 C 预处理指令。

MACRO_EXPANSION
若为 YES，doxygen 将在源代码中展开所有的宏定义。若为 NO（默认），只会执行条件编译。宏展开的另一种控制方法是将 EXPAND_ONLY_PREDEF 设为 YES。

EXPAND_ONLY_PREDEF
若此标记和 MACRO_EXPANSION 均设为 YES，宏展开将受限于宏所指定的 PREDEFINED，EXPAND_AS_DEFINED 标记。

SEARCH_INCLUDES
若为 YES（默认），如果#include 被找到，那么将搜索 INCLUDE_PATH（下一条）中的头文件。

INCLUDE_PATH
指定一个或多个目录，用于预处理在输入文件未找到包含的头文件时将搜索的其他位置。

PREDEFINED
在预处理运行之前，指定一个或多个宏定义名（类似于运行 gcc -D）。此标记的参数是一个宏列表，如下：name 或者 name=definition（无空格）。如果 definition 和“=”被省略，则假定“=1”.用来防止一个宏定义来自于，使用了#undef 的非受限标识 undefined，或是在递归展开使用：=操作符而不是=操作符。

EXPAND_AS_DEFINED
若 MACRO_EXPANSION， EXPAND_ONLY_PREDEF 均为 YES，此标记用于指定一个将被展开的宏名列表。并将源码中查询列表中的宏定义。如果你想使用一个不同的宏定义，可用 PREDEFINED 标记。

SKIP_FUNCTION_MACROS

若为 YES（默认），所有风格类似于函数，且处于单独的一行内使用一个全部大写的名字并没有以分号结束的宏，将被 doxygen 的预处理删除。为了代码的风格一致，通常会使用函数宏，如果不将其移除，将会引起解析器的混乱。

与外部引用相关的选项

TAGFILES
用于指定一个或多个 tag 文件。
查看“doxytag 用法”小节以获得 tag 文件用法的更多信息。

可每一个 tag 文件选定一个将添加的外部文档的初始位置，tag 文件的格式中不附带位置信息的方式，如下：
TAGFILES = file1 file2 ...

在 tag 文件添加位置信息方式，如下：
TAGFILES = file1=loc1 "file2 = loc2" ...

loc1，loc2 可以是相对路径，绝对路径或者 URL，如果 tag 文件出现一个位置信息，Installdox 工具（查看“Installdox 用法”小节获得更多信息）无需运行来修正这个链接。

注意：每个 tag 文件都有一个唯一的名称（此名称不包含路径），当 doxygen 运行后在目录无法定位一个 tag 文件时，你必须指定一个路径给这个 tag 文件。

GENERATE_TAGFILE
当在此标记后指定一个文件名时，doxygen 将创建一个 tag 文件，基于 doxygen 所读取的输入文件。查看“doxytag 用法”小节以获得 tag 文件用法的更多信息。

ALLEXTERNALS
若为 YES，在类的索引中将列表所有的外部类，若为 NO，只会列表外部继承类。

EXTERNAL_GROUPS
若为 YES，在模块索引中将列表所有的外部组，若为 NO，只会列表当前项目的组。

PERL_PATH
指定 perl 脚本翻译器的绝对路径和程序名（等同于 which perl 的运行结果）。

与 Dot 相关的选项

CLASS_DIAGRAMS

若为 YES（默认），doxygen 将为基类或超类生成一个类图表（在 HTML 和 latex 中）。若为 NO 将取消类图表。注意，此选项将被下面的 HAVE_DOT 选项所取代，它只是一个备用选择。推荐安装 dot，因为使用它能产生更丰富的图形效果。

MSCGEN_PATH

可使用\msc 命令在 doxygen 的注释中定义消息序列图。Doxygen 将运行 mscgen 工具(http://www.mcternan.me.uk/mscgen/）生成此图并插入到文档中。此选项允许你指定 mscgen 工具的保存目录，若选项后留空，将在默认搜索路径中查找该工具。

HAVE_DOT

若为 YES，doxygen 将假定在系统路径中 dot 工具是有效的，此工具是 GraphViz（一个来自 AT&T 和朗讯贝尔实验室的图形可视化工具包）的一部分，若此选项为 NO（默认），那么该小节中其他选项将全部失效。

DOT_NUM_THREADS

指定 doxygen 能允许并行调用 dot 的线程数。当设为 0（默认）doxygen 将依据系统中有效的处理器个数设定 dot 的线程数。你可以设定一个大于 0 的值，用于控制 CPU 载荷和处理速度之间的平衡。

DOT_FONTNAME

默认情况下，doxygen 将写入一个名为 FreeSans.ttf 字体文件到输出目录中，并在 doxygen 生成的所有 dot 文件中引用。但这个字体并不包含一切可能的 unicode 字符，所以当你需要这些无法显示的字符时（或者更换一种不同的字体），你可以用 DOT_FONTNAME 指定字体的名称。你需要确认 dot 能否找到指定的字体，可将字体放置到一个标准的位置，或是配置 DOTFONTPATH 环境变量，又或者在 DOT_FONTPATH 指定的目录中包含该字体。

DOT_FONTSIZE

设定 dot 图表中字体的尺寸，默认为 10pt。

DOT_FONTPATH

默认情况下，doxygen 将告知 dot 从输出目录查找 FreeSans.ttf 字体文件（或是其他 doxygen 放置字体文件的目录），如果使用 DOT_FONTNAME 指定了一个特殊字体，你要在此选项后设定相应的路径使得 dot 能够找到。

CLASS_GRAPH

若 CLASS_GRAPH，HAVE_DOT 都为 YES，doxygen 将为每个文档化的类生成一个类图，显示直接和间接的继承关系。设定该标记为 YES 后将导致 CLASS_DIAGRAMS 标记为 NO。

COLLABORATION_GRAPH

若 COLLABORATION_GRAPH，HAVE_DOT 都为 YES，doxygen 将为每个文档化的类生成一个协作图，显示直接和间接的类与其他文档化类的执行依赖关系（继承，包容，类引用变量）。

GROUP_GRAPHS

若 GROUP_GRAPHS，HAVE_DOT 都为 YES，doxygen 将成一个组图，显示直接的组的依赖关系。

UML_LOOK

若为 YES，doxygen 将生成继承和协作的图表，在样式类似于 OMG 的统一建模语言。

TEMPLATE_RELATIONS

若 TEMPLATE_RELATIONS，HAVE_DOT 都为 YES，doxygen 将显示出模板和它的实例化之间的关系。

HIDE_UNDOC_RELATIONS

若为 YES，如果目标未被文档化或者不是一个类，那么继承和协作图中将隐藏继承和调用的关系。

INCLUDE_GRAPH

若 ENABLE_PREPROCESSING, SEARCH_INCLUDES, INCLUDED_ GRAPH，HAVE_DOT 都为 YES，doxygen 将为每个文档化文件生成一个包容图，显示与其他文件的直接和间接的包容依赖关系。

INCLUDED_BY_GRAPH

若 ENABLE_PREPROCESSING, SEARCH_INCLUDES, INCLUDED_BY_GRAPH，HAVE_DOT 都为 YES，doxygen 将为每个文档化的头文件，生成一个包容图以显示那些文件直接或间接包含过此头文件。

CALL_GRAPH

若 CALL_GRAPH，HAVE_DOT 都为 YES，doxygen 将被每一个全局函数或类方法生成一个调用依赖图，注意，有效该选项将会显著增加运行的时间，但大多数情况下，它优于只在所选函数中使用\callgraph 命令来有效调用图的生成。

CALLER_GRAPH

若 CALLER_GRAPH，HAVE_DOT 都为 YES，doxygen 将被每一个全局函数或类方法生成一个被调用的依赖图，注意，有效该选项将会显著增加运行的时间，但大多数情况下，它优于只在所选函数中使用\callergraph 命令来有效被调用图的生成。

GRAPHICAL_HIERARCHY

若 GRAPHICAL_HIERARCHY，HAVE_DOT 都为 YES，doxygen 将为所有的类生成层次图，以替换原有的纯文字描述。

DIRECTORY_GRAPH

若 DIRECTORY_GRAPH，HAVE_DOT 都为 YES，doxygen 将使用图形的方式展现出目录之间的依赖关系，这种依赖关系可以确定目录中文件之间的包含关系。

DOT_GRAPH_MAX_NODES

用于设定在图中将显示的最大节点数。若图中的节点数大于该值，doxygen 将对图进行处理，多余的节点将被放置到图中可见的类似于一个红盒子的节点内。注意，如果图中根节点的下级子节点数已经超过该选项的设定值，doxygen 将无法显示所有的节点，图的尺寸也能通过 MAX_DOT_GRAPH_DEPTH 进行限定。

MAX_DOT_GRAPH_DEPTH

用于设定 dot 工具生成的图中显示的最大深度，深度值为 3，意味着根结点下的每条路径上最多只能获得 3 层节点的显示。根结点的索引值为 0，即计算深度值时它将被忽略。注意，对于大型代码库来说，设定此选项为 1 或 2 能大大减稍所需的计算时间，图的尺寸也能通过 DOT_GRAPH_MAX_NODES 进行限定，选定深度值为 0（默认）则表示无深度限制。

DOT_IMAGE_FORMAT

用于设定 dot 工具生成的图片格式，它有可能是 gif， jpg， png，如果选项后留空则使用 png 格式。

DOT_PATH
指定 dot 工具放置的位置，若选项后留空，则假定 dot 工具处于能被查找到的路径之下。

DOTFILE_DIRS
用于指定文档中包含一个或多个保存有 dot 文件的目录（查看\dotfile 命令）

DOT_TRANSPARENT
若为 YES，将生成背景透明的图片。该选项默认状态下是无效的，因为在 windows 中 dot 并不支持这个功能。警告：与使用的平台有关，有效该选项可能会导致在图形的边沿出现抗锯齿的印记（它们将变得难以阅读）。

DOT_MULTI_TARGETS
若为 YES，允许 dot 在一次运行中生成多个输出文件（在命令行中使用 -o 和 -T 的选项），这将使得 dot 运行得更快，但只有 dot 新版本（>1.8.10）才支持，此特性在默认情况下是禁用的。

GENERATE_LEGEND
若为 YES（默认），doxygen 将生成一个说明页面，用于解释在 dot 生成的图中多种方框和箭头的所表达的意义。

DOT_CLEANUP
若为 YES（默认），doxygen 将删除在生成多种图时使用的 dot 临时文件。

例子

假定你已有了一个包含两个文件的简单项目：一个 example.cc 源文件和一个 example.h 头文件，以及一个同样简单的最小配置文件：

```
INPUT            = example.cc example.h
```

假定例子使用 Qt 类，perl 位于/usr/bin，一个更实用的配置文件，如下：

```
PROJECT_NAME     = Example
INPUT            = example.cc example.h
WARNINGS         = YES
TAGFILES         = qt.tag
PERL_PATH        = /usr/bin/perl
SEARCHENGINE     = NO
```

以下是为生成 QdbtTabular（http://www.stack.nl/~dimitri/qdbttabular/index.html）程序包文档所用的配置文件：

```
PROJECT_NAME     = QdbtTabular
OUTPUT_DIRECTORY = html
WARNINGS         = YES
INPUT            = examples/examples.doc src
```

```
FILE_PATTERNS     = *.cc *.h
INCLUDE_PATH      = examples
TAGFILES          = qt.tag
PERL_PATH         = /usr/local/bin/perl
SEARCHENGINE      = YES
```

以下是为重新生成 Qt-1.44 源码文档所用的配置文件：

```
PROJECT_NAME          = Qt
OUTPUT_DIRECTORY      = qt_docs
HIDE_UNDOC_MEMBERS    = YES
HIDE_UNDOC_CLASSES    = YES
ENABLE_PREPROCESSING = YES
MACRO_EXPANSION       = YES
EXPAND_ONLY_PREDEF    = YES
SEARCH_INCLUDES       = YES
FULL_PATH_NAMES       = YES
STRIP_FROM_PATH       = $(QTDIR)/
PREDEFINED            = USE_TEMPLATECLASS Q_EXPORT= \
                        QArrayT:=QArray \
                        QListT:=QList \
                        QDictT:=QDict \
                        QQueueT:=QQueue \
                        QVectorT:=QVector \
                        QPtrDictT:=QPtrDict \
                        QIntDictT:=QIntDict \
                        QStackT:=QStack \
                        QDictIteratorT:=QDictIterator \
                        QListIteratorT:=QListIterator \
                        QCacheT:=QCache \
                        QCacheIteratorT:=QCacheIterator \
                        QIntCacheT:=QIntCache \
                        QIntCacheIteratorT:=QIntCacheIterator \
                        QIntDictIteratorT:=QIntDictIterator \
                        QPtrDictIteratorT:=QPtrDictIterator
INPUT                 = $(QTDIR)/doc \
                        $(QTDIR)/src/widgets \
                        $(QTDIR)/src/kernel \
                        $(QTDIR)/src/dialogs \
                        $(QTDIR)/src/tools
FILE_PATTERNS         = *.cpp *.h q*.doc
INCLUDE_PATH          = $(QTDIR)/include
RECURSIVE             = YES
```

以下是用于 Qt-2.1 的推荐设置：

```
PROJECT_NAME           = Qt
PROJECT_NUMBER         = 2.1
HIDE_UNDOC_MEMBERS     = YES
HIDE_UNDOC_CLASSES     = YES
SOURCE_BROWSER         = YES
INPUT                  = $(QTDIR)/src
FILE_PATTERNS          = *.cpp *.h q*.doc
RECURSIVE              = YES
EXCLUDE_PATTERNS       = *codec.cpp moc_* */compat/* */3rdparty/*
ALPHABETICAL_INDEX     = YES
COLS_IN_ALPHA_INDEX    = 3
IGNORE_PREFIX          = Q
ENABLE_PREPROCESSING   = YES
MACRO_EXPANSION        = YES
INCLUDE_PATH           = $(QTDIR)/include
PREDEFINED             = Q_PROPERTY(x)= \
                         Q_OVERRIDE(x)= \
                         Q_EXPORT= \
                         Q_ENUMS(x)= \
                         "QT_STATIC_CONST=static const " \
                         _WS_X11_ \
                         INCLUDE_MENUITEM_DEF
EXPAND_ONLY_PREDEF     = YES
EXPAND_AS_DEFINED      = Q_OBJECT_FAKE Q_OBJECT ACTIVATE_SIGNAL_WITH_PARAM \
                         Q_VARIANT_AS
```

doxygen 预处理可代用通常情况下的 C 预处理器，但无法实现宏的完全展开。

特 殊 命 令 字

简介

在文档中所有的命令字都是有一个反斜杠“\”或是一个 at 符号“@”开始的，如果你愿意可以将所有以“\”开头部分替换成“@”.

一部分命令会有一个或多个参数，每个参数都有一定的位置区域：

- 参数使用<sharp>括号，表示单个字。
- 参数使用(round)括号，表示将延伸到命令所在行的末尾。
- 参数使用{curly}括号，表示将延伸到下一个段落，本段落将受限于一个空白行或是一个区域指示符。

参数使用[square]括号则表示它是可选的。

这有一个按字母顺序排列的所有命令的列表，并已引用到它们的文档中。
（因为文档尺寸的问题而省略）

下面的段落提供了所有被确认的 doxygen 命令列表，无法识别的命令被被视为普通文本。

--- 结 构 化 指 示 ---

\addtogroup <name> [(title)]

使用\defgroup 同样可以定义一个组，但相比之下，多次使用相同<name>的\addtogroup 命令并不会出现一个警告，而它是一个将命令中附带的 title 与文档进行合并的组。but rather one group with a merged documentation and the first title found in any of the commands

title 是可选的，所以该命令可使用“@{”和“@}”，添加一定数量的实体到一个已存在的组中，如：

```
/*! \addtogroup mygrp
   *   Additional documentation for group `mygrp'
   *   @{
   */

  /*!
   *   A function
   */
```

```
void func1()
{
}

/*! Another function */
void func2()
{
}

/*! @} */
```

也可查看“组合”章节，\defgroup，\ingroup，\weakgroup 命令。

\callgraph

当该命令放置到一个函数或方法的注释块中，且 HAVE_DOT 设为 YES，doxygen 将被函数创建一个调用图（提供函数或方法调用其他文档函数的执行顺序），调用图的生成与否将不会参考 CALL_GRAPH 的设定值。
注意，调用图的完整性（和正确性）都依赖于 doxygen 代码解析器的能力。

\callergraph

当该命令放置到一个函数或方法的注释块中，且 HAVE_DOT 设为 YES，doxygen 将被函数创建一个调用者图（提供函数或方法被其他文档函数调用的执行顺序），调用图的生成与否将不会参考 CALL_GRAPH 的设定值。
注意，调用图的完整性（和正确性）都依赖于 doxygen 代码解析器的能力。

\category <name> [<header-file>] [<header-name>]

只适用于 Object-C：为一个名为<name>的类指定一个文档注释块，此命令参数应与\class 命令相同。
也可查看\class 命令。

\class <name> [<header-file>] [<header-name>]

为一个名为<name>类指定一个文档注释块，可选择指定的一个头文件和一个头名称。如果指定了一个头文件，在 HTML 文档中将包含一个指向此头文件的完全副本的链接。<header-name>参数可用于覆盖<header-file>之外的类文档中所使用链接名称，如果在默认包含路径中无法定位包含文件（如<X11/X.h>）时，<header-name>就非常有用。你也可以指定<header-name>，使用引号或尖括号中加入头名，使之看起来象是包容语句。With the <header-name> argument you can also specify how the include statement should look like, by adding either quotes or sharp brackets around the name 尖括号可用于已给出的头名。注意，后两个参数也可由\headerfile 命令来指定。

例子：

```
/* A dummy class */

class Test
{
};

/*! \class Test class.h "inc/class.h"
 *   \brief This is a test class.
 *
 * Some details about the Test class
 */
```

点击此处获得 doxygen 生成 HTML 文档的方法。

\def <name>

为一个#define 宏指定一个文档注释块。

例子：

```
/*! \file define.h
      \brief testing defines

      This is to test the documentation of defines.
*/

/*!
   \def MAX(x,y)
   Computes the maximum of \a x and \a y.
*/

/*!
    Computes the absolute value of its argument \a x.
*/
#define ABS(x) (((x)>0)?(x):-(x))
#define MAX(x,y) ((x)>(y)?(x):(y))
#define MIN(x,y) ((x)>(y)?(y):(x))
          /*!< Computes the minimum of \a x and \a y. */
```

点击此处获得 doxygen 生成 HTML 文档的方法。

\defgroup <name> (group title)

为了类，文件或名字空间的组，指定一个文档注释块，它可用于类，文件，名字空间和文档的其他类别进行归类，你也可将组作为其他组的成员，这样就能创建出一个组的层次。

<name>是一个单字的标识符。

也可查看“组合”章节，\defgroup，\ingroup，\weakgroup 命令。

\dir [<path fragment>]

为了一个目录指定一个文档注释块，path fragment 参数将会包含项目中的目录名和唯一 w.r.t.的其他目录的完整路径。The "path fragment" argument should include the directory name and enough of the path to be unique w.r.t. the other directories in the project. SHOW_DIRECTORIES 选项将确定目录信息是否被显示，而 STRIP_FROM_PATH 选项将确定目录出现在输出之前，是否要从完整的路径中剥离出来。

\enum <name>

为一个名为<name>的枚举类型指定一个文档注释块，如果此枚举是类中的一个成员，且文档块处于类定义之外，类的作用域同样需要指定。如果一个注释块位于枚举声明之前，那么\enum 注释可能被省略。

注意，一个无名的枚举类型将不能被文档化，但一个无名的枚举值是可以使用的。

例子：

```
class Test
{
  public:
    enum TEnum { Val1, Val2 };

    /*! Another enum, with inline docs */
    enum AnotherEnum
    {
      V1, /*!< value 1 */
      V2   /*!< value 2 */
    };
};

/*! \class Test
 * The class description.
 */

/*! \enum Test::TEnum
 * A description of the enum type.
 */

/*! \var Test::TEnum Test::Val1
 * The description of the first enum value.
 */
```

点击此处获得 doxygen 生成 HTML 文档的方法。

\example <file-name>

为一个源码的例子指定一个文档注释块，<file-name>是源码文件名，在注释块包含的文档之后，将包含此文件的文本。所有的例子将被放入一个列表中，为了已文档化的成员和类，与文档之间的交叉引用，将对源码进行扫描，源文件或目录将在 doxygen 的配置文件中使用 EXAMPLE_PATH 标记指定。

如果<file-name>也就是在 EXAMPLE_PATH 标记指定的例子文件名，并非是唯一的，你需要包含它的绝对路径，以消除它的二义性。

如果例子中需要多个源文件，可以是用\include 命令

例子：

```
/** A Test class.
 *   More details about this class.
 */

class Test
{
  public:
    /** An example member function.
     *   More details about this function.
     */
    void example();
};
```

```
void Test::example() {}

/** \example example_test.cpp
 * This is an example of how to use the Test class.
 * More details about this example.
 */
```

以下是 example_test.cpp 例子文件：

```
void main()
{
   Test t;
   t.example();
}
```

点击此处获得 doxygen 生成 HTML 文档的方法。
也可查看\include 命令

\extends <name>

用于手动指定一个继承关系，当编程语言不支持这个特性时（如 C）。

在例子目录中 manual.c 文件将显示如何使用这个命令。

点击此处获得 doxygen 生成 HTML 文档的方法。
也可查看\implements，\memberof 命令

\file [<name>]

为一个名为<name>的源文件或头文件，指定一个文档注释块。如果文件名本身并不是唯一的，那么这个文件名可能包含（部分包含）了它的路径，如果该文件名被省略（如\file 后留空），那么包含\file 命令的文档块将属于它已定位到的那个文件。

要点：全局函数，变量，类型转换，枚举的文档只能包含在输出中，并且它们也已被文档化。

例子：

```
/** \file file.h
 * A brief file description.
 * A more elaborated file description.
 */

/**
 * A global integer value.
 * More details about this value.
 */
extern int globalValue;
```

点击此处获得 doxygen 生成 HTML 文档的方法。

注意，在上面的例子中 JAVADOC_AUTOBRIEF 已设为 YES。

\fn (function declaration)

为一个函数（既可以是全局函数也可以是一个类的成员函数）指定一个文档注释块，此命令只能工作于，一个注释块未被放置到函数声明或定义的前面或后面。

如果注释块位于函数声明或定义的前面，此命令可被省略（为了避免重复）。

在一个单行\fn 命令之后，才有一个包含指定参数的完整函数定义，且参数止于行尾。

警告：在此命令并非绝对需要的情况下不要使用它，因为它将导致信息重复而出现错误。

例子：

```
class Test
{
  public:
    const char *member(char,int) throw(std::out_of_range);
};

const char *Test::member(char c,int n) throw(std::out_of_range) {}

/*! \class Test
 * \brief Test class.
 *
 * Details about Test.
 */

/*! \fn const char *Test::member(char c,int n)
 *   \brief A member function.
 *   \param c a character.
 *   \param n an integer.
 *   \exception std::out_of_range parameter is out of range.
 *   \return a character pointer.
 */
```

点击此处获得 doxygen 生成 HTML 文档的方法。
也可查看\var，\typedef 命令

\headerfile <header-file> [<header-name>]

打算用于类，结构，联合的文档，且这些文档都位于它们定义的前面。此命令的参数与\cmdclass 命令的第二，三个参数相同。header-file 名称会引用一个文件，它能被应用程序为获取类，结构，联合的定义而包含。<header-name>参数可用于覆盖<header-file>之外的类文档中所使用链接名称，如果在默认包含路径中无法定位包含文件（如<X11/X.h>）时，<header-name>就非常有用。

你也可以指定<header-name>，使用引号或尖括号中加入头名，使之看起来象是包容语句。With the <header-name> argument you can also specify how the include statement should look like, by adding either quotes or sharp brackets around the name 尖括号可用于已给出的头名。

若<header-file>，<header-name>都使用双引号，当前文件（该命令查到的那个文件）将被引用。为了将一个\headerfile 命令的注释块插入到 test.h 文件中，以下有三种等价的命令操控方式：

```
\headerfile test.h "test.h"
\headerfile test.h ""
\headerfile ""
```

使用尖括号时你无需指定目标，如果你希望明确地指出，可也以使用以下方式：

```
\headerfile test.h <test.h>
\headerfile test.h <>
\headerfile <>
```

希望将全局默认的包含表述用于本地的包含，可设定 FORCE_LOCAL_INCLUDES 为 YES。
希望完全取消包含信息的显示，可设定 SHOW_INCLUDE_FILES 为 NO。

\hideinitializer

一个定义的缺省值和一个变量的初始化都将显示在文档中，除非它们的行数超过 30.放置该命令在一个定义或变量的注释块中，初始化将一直被隐藏。
查看\showinitializer 命令

\implements <name>

用于手动指定一个继承关系，当编程语言不支持这个特性时（如 C）。

在例子目录中 manual.c 文件将显示如何使用这个命令。

点击此处获得 doxygen 生成 HTML 文档的方法。
也可查看\implements，\memberof 命令

\ingroup (<groupname> [<groupname> <groupname>])

若该命令被放置到一个类，文件，名字空间的注释块中，它将添加一个<groupname>的组或是组 id。
查看“组合”一章，\defgroup，\addtogroup，\weakgroup 命令

\interface <name> [<header-file>] [<header-name>]

为一个名为<name>的接口指定一个文档注释块，参数与\class 命令相同。
查看\class 命令

\internal

该命令将写入一个消息（`For internal use only'）到标准输出 stdout 中，在此命令之后的所有文本，直到注释块结束或是小节的末尾，都将被标记成“internal”

如果该命令被放置的一个小节中（查看\section 中的例子），那么命令后的所有子小节将被看成 internal，只有与其在同一层级的新节才能重新可见。

你也可使用配置文件的 INTERNAL_DOCS 设定显示或隐藏内部文档。

\mainpage [(title)]

如果该命令被放置到一个注释块，此块将被用于定制 HTML 的索引页，或 latex 的首章。

title参数是可选的,并能替换掉doxygen生成的默认标题。如果你不需要任何标题,可以在命令后指定notitle。

这有一个例子：

```
/*! \mainpage My Personal Index Page
 *
 * \section intro_sec Introduction
 *
 * This is the introduction.
 *
 * \section install_sec Installation
 *
 * \subsection step1 Step 1: Opening the box
 *
 * etc...
 */
```

也可以使用\ref 对主页进行引用（如果树型浏览无效，你也可以使用\ref）。
查看\section,\subsection,\page 命令

\memberof <name>

该命令使得类中的一个函数与一个成员之前已不存在差异，如同\relates 命令所做的。memberof 命令中的函数只能被表述成一个类的真实成员，当编程语言不支持成员函数的概念时它将非常有用（如 C）。

有可能与该命令一同被使用的有：\public, \protected ，\private。

例子目录中的 manal.c 文件将展示如何使用该命令。
点击此处获得 doxygen 生成 HTML 文档的方法。
也可查看\extends, \implements, \public, \protected，\private 命令。

\name (header)

该命令可将成员组中的头定义转变成一个注释块，注释块将位于一个包含成员组的//@{ ... //@}块。

查看“成员组”章节获取一个例子。

\namespace <name>

为一个名为<name>的名字空间指定一个文档注释块

\nosubgrouping

可放置在一个类文档中，用于在合并成员组时，避免 doxygen 放置一个公有，保护或私有的子分组。

\overload [(function declaration)]

用于为一个重载成员函数生成后续的标准文本：

这是一个为了便利而提供的重载函数,它不同于上述函数,只包含那些它接受的参数。It differs from the above function only in what argument(s) it accepts

如果在函数声明或定义之前，无法定位重载成员函数的文档，能用可选参数来指定正确的函数。

在生成的消息之后将会附带文档块内其他的文档。Any other documentation that is inside the documentation block will by appended after the generated message

注意 1：你要承担起确认文档化的重载成员正确与否的责任，在这种情况下为了防止文档重新排列，必须将 SORT_MEMBER_DOCS 设为 NO。

注意 2：在一行注释中\overload 命令无法工作。

例子：

```
class Test
{
   public:
      void drawRect(int,int,int,int);
      void drawRect(const Rect &r);
};

void Test::drawRect(int x,int y,int w,int h) {}
void Test::drawRect(const Rect &r) {}

/*! \class Test
 *   \brief A short description.
 *
 *   More text.
 */

/*! \fn void Test::drawRect(int x,int y,int w,int h)
 * This command draws a rectangle with a left upper corner at ( \a x , \a y ),
 * width \a w and height \a h.
 */

/*!
 * \overload void Test::drawRect(const Rect &r)
 */
```

点击此处获得 doxygen 生成 HTML 文档的方法。

\package <name>

为一个名为<name>的 Java 包指定一个文档注释块。

\page <name> (title)

指定一个注释块，它包含了一个与特定的类，文件，成员非直接关联的文档块。HTML 生成器将创建一个包含此文档的页面，而 latex 生成器将在`页文档`的章节中开始一个新的小节。

例子：

```
/*! \page page1 A documentation page
  Leading text.
```

```
   \section sec An example section
   This page contains the subsections \ref subsection1 and \ref subsection2.
   For more info see page \ref page2.
   \subsection subsection1 The first subsection
   Text.
   \subsection subsection2 The second subsection
   More text.
*/

/*! \page page2 Another page
   Even more info.
*/
```

点击此处获得 doxygen 生成 HTML 文档的方法。

注意：<name>参数会包含字母和数字，如果你希望在<name>参数中使用大写字母（如 MYPAGE1），或是大小写混用（如 MyPage1），可以设定 CASE_SENSE_NAMES 为 YES。如果你的文件系统对大小写敏感，这样做是明智的。另外（为获得更好的移植性）你也可在页面内所有<name>的引用中全部使用小写字母（如 mypage1）。

也可查看\section,\subsection,\ref 命令

\private

指定注释块中的私有文档化成员，它只能被同一类的其他成员访问。

注意，doxygen 会自动确认面对对象语言中的成员保护层级，此命令用于不支持保护层级的概念的语言。（如 C，PHP4）

私有成员小节的开始处，有一种与 C++中“private：”类似的标记方法，使用\privatesection

也可查看\memberof, \public, \protected 命令

\property (qualified property name)

为一个属性（即可是全局也可是一个类成员）指定一个文档注释块，此命令等价于\var，\fn。

也可查看\var，\fn 命令。

\protected

指定注释块中的保护文档化成员，它只能被同一类的其他成员或是派生类访问。

注意，doxygen 会自动确认面对对象语言中的成员保护层级，此命令用于不支持保护层级的概念的语言。（如 C，PHP4）

保护成员小节的开始处，有一种与 C++中“protected：”类似的标记方法，使用\protectedsection

也可查看\memberof, \public, \private 命令

\protocol <name> [<header-file>] [<header-name>]

为 Object-C 中名为<name>的协议指定一个文档注释块，它的参数与\class 相同。

也可查看\class 命令

\public

指定注释块中的公有文档化成员，它能被其他类或是函数访问。

注意，doxygen 会自动确认面对对象语言中的成员保护层级，此命令用于不支持保护层级的概念的语言。（如 C，PHP4）

公有成员小节的开始处，有一种与 C++中“public：”类似的标记方法，使用\publicsection

也可查看\memberof, \protected, \private 命令

\relates <name>

用于一个名为<name>非成员函数的文档，它可以放置这个函数到类文档的`关联函数`小节内。该命令对于非友元函数与类之间建立强耦合是非常有用的。它无法用于文件，只能用于函数。

例子：

```
/*!
 * A String class.
 */

class String
{
   friend int strcmp(const String &,const String &);
};

/*!
 * Compares two strings.
 */

int strcmp(const String &s1,const String &s2)
{
}

/*! \relates String
 * A string debug function.
 */
void stringDebug()
{
}
```

点击此处获得 doxygen 生成 HTML 文档的方法。

\relatesalso <name>

用于一个名为<name>非成员函数的文档，它可以放置这个函数到类文档的`关联函数`小节内，也可是远离它正常文档的位置。该命令对于非友元函数与类之间建立强耦合是非常有用的。只能用于函数。

\showinitializer

只显示定义的默认值和变量的初始化，如果它们的文本行少于 30，在变量或定义的注释块中放置这个命令，初始化过程将无条件被显示出来。

也可查看命令\hideinitializer

\struct <name> [<header-file>] [<header-name>]
为一个名为 name 的结构指定一个文档注释块，它的参数与\class 相同
也可查看命令\class。

\typedef (typedef declaration)
为一个类型转换（既可是全局也可是一个类成员）指定一个文档注释块，此命令等价于\var，\fn。
也可查看命令\var，\fn。

\union <name> [<header-file>] [<header-name>]
为一个名为 name 的联合指定一个文档注释块，它的参数与\class 相同
也可查看命令\class。

\var (variable declaration)
为一个变量或枚举值（既可是全局也可是一个类成员）指定一个文档注释块，此命令等价于\typedef，\fn
也可查看命令\typedef，\fn。

\weakgroup <name> [(title)]
用法与\addtogroup 相似，但当解决组定义冲突时，它将会有一个更低的优先级。
也可查看“组合”章节和命令\addtogroup。

--- 小节指示命令 ---

\attention { attention text }
开始一个可输入需处理信息的段落，此段落将会缩排，段落的文本不会指定一个内部结构，所有视觉增强的命令可以放置到该段落中，多个相邻的\attention 命令将被组合到一个单独的段落中，当遇到一个空白行或是其他的小节命令，那么\attention 将终止。

\author { list of authors }
开始一个可输入一个或多个作者名的段落，此段落将会缩排，段落的文本不会指定一个内部结构，所有视觉增强的命令可以放置到该段落中，多个相邻的\author 命令将被组合到一个单独的段落中，每个作者描述都将开启一个新行，另外一个\author 命令可能会有若干个作者，当遇到一个空白行或是其他的小节命令，那么\author 将终止。

例子：

```
/*! \class WindowsNT
 *  \brief Windows Nice Try.
 *  \author Bill Gates
```

```
 *   \author Several species of small furry animals gathered together
 *                in a cave and grooving with a pict.
 *   \version 4.0
 *   \date       1996-1998
 *   \bug It crashes a lot and requires huge amounts of memory.
 *   \bug The class introduces the more bugs, the longer it is used.
 *   \warning This class may explode in your face.
 *   \warning If you inherit anything from this class, you're doomed.
 */

class WindowsNT {};
```

点击此处获得 doxygen 生成 HTML 文档的方法。

\brief {brief description}

开始一个作为简明描述的段落，在文档页的起始部分可使用类和文件的简明描述的列表，可在细节描述的前端和成员声明处放置类成员和文件成员的简明描述。一个简明描述可能需要若干行(尽管建议它保持简明的风格)。当遇到一个空白行或是其他的小节命令，那么简明描述将终止。如果出现多个\brief 命令，那么它们将被组合在一起。查看命令\author 的例子。

等同于命令\short

\bug { bug description }

开始一个可报告一个或多个 bug 的段落，此段落将会缩排，段落的文本不会指定一个内部结构，所有视觉增强的命令可以放置到该段落中，多个相邻的\bug 命令将被组合到一个单独的段落中，每个 bug 都将开启一个新行，另外一个\bug 命令可能会有若干个 bug，当遇到一个空白行或是其他的小节命令，那么\bug 将终止。查看命令\author 的例子。

\cond [<section-label>]

开始一个带条件的小节，使用命令\endcond 来终止该小节，通常在其他的注释块中可查找到\endcond。此对命令的目的是，（条件化）在处理中将一部分文件排除。（老版本的 doxygen 只能使用 C 的预处理命令来实现）

小节将包含在命令\cond，\endcond 之间，并将小节的标号加入到配置选项 ENABLED_SECTIONS 中，如果标号被忽略，该小节也将在处理中被无条件忽略。

在注释块中的条件小节，可使用一个\if … \endif 块。

条件小节能被嵌套，在这种情况下，如果它和嵌套的小节都被包含在其中，那么只有被嵌套的小节才会显示。

这有一个例子显示了该命令的使用：

```
/** An interface */
class Intf
{
  public:
    /** A method */
    virtual void func() = 0;

    /// @cond TEST
```

```
    /** A method used for testing */
    virtual void test() = 0;

    /// @endcond
};

/// @cond DEV
/*
 *   The implementation of the interface
 */
class Implementation : public Intf
{
  public:
    void func();

    /// @cond TEST
    void test();
    /// @endcond

    /// @cond
    /** This method is obsolete and does
     *   not show up in the documentation.
     */
    void obsolete();
    /// @endcond
};

/// @endcond
```

输出的差异将依赖于 ENABLED_SECTIONS 是包含 TEST 或是 DEV。

\date { date description }

开始一个可输入一个或多个日期的段落，此段落将会缩排，段落的文本不会指定一个内部结构，所有视觉增强的命令可以放置到该段落中，多个相邻的\date 命令将被组合到一个单独的段落中，每个 date 都将开启一个新行，另外一个\date 命令可能会有若干个日期，当遇到一个空白行或是其他的小节命令，那么\bug 将终止。查看命令\author 的例子。

\deprecated { description }

开始一个指示文档块是否弃用的段落，可用于描述替代方式，预定生命期等等。

\details {detailed decription}

如同\brief 开始一个简明描述，\details 将开始一个细节描述，你也可以开始一个新段落（加入空白行），而无需加入命令\details。

\else

开始一个条件小节，如果前一个条件小节无效。前一个条件小节可使用命令\if, \ifnot, \elseif 开始。
也可查看命令\if, \ifnot, \elseif。

\elseif <section-label>

开始一个条件文档小节，如果前一个条件小节无效。当条件小节默认为无效时，你必须放置该命令的 section-label 到配置标记 ENABLED_SECTIONS 中使得命令\elseif 有效。条件块可以嵌套，如果所有嵌套的条件都有效，那么也只有最内层的嵌套小节才会被读取。
也可查看命令\endif, \ifnot, \else, \elseif。

\endcond

结束一个由\cond 开始的条件小节。
也可查看命令\cond

\endif

结束一个由\if 或\ifnot 开始的条件小节，每一个\if 或\ifnot 只能一一对应地匹配之后的\endif。
也可查看命令\if，\ifnot

\exception <exception-object> { exception description }

为一个名为<exception-object>异常对象而开始一个描述，其次是此异常的描述。不会去检查是否存在这个异常对象。段落的文本不会指定一个内部结构，所有视觉增强的命令可以放置到该段落中，多个相邻的 \exception 命令将被组合到一个单独的段落中，每个参数的描述都将开启一个新行，当遇到一个空白行或是其他的小节命令，那么\bug 将终止。查看命令\fn 的例子。

注意：与\exceptions 同义

\if <section-label>

开始一个条件化的文档小节，并使用一个匹配的命令\endif 结束。当条件小节默认为无效时，你必须放置该命令的 section-label 到配置标记 ENABLED_SECTIONS 中使得命令\if 有效。条件块可以嵌套，如果所有嵌套的条件都有效，那么也只有最内层的嵌套小节才会被读取。

例子：

```
/*! Unconditionally shown documentation.
    *   \if Cond1
    *     Only included if Cond1 is set.
    *   \endif
    *   \if Cond2
    *     Only included if Cond2 is set.
    *     \if Cond3
    *       Only included if Cond2 and Cond3 are set.
    *     \endif
    *     More text.
    *   \endif
    *   Unconditional text.
    */
```

如果你在 alias 中使用了条件命令，如果在两种语言中文档化一个类，你可以这样使用：

例子 2：

```
/*! \english
 *   This is English.
 *   \endenglish
```

```
 *   \dutch
 *   Dit is Nederlands.
 *   \enddutch
 */
class Example
{
};
```

以下是配置文件中定义的 alias：

```
ALIASES    = "english=\if english" \
             "endenglish=\endif" \
             "dutch=\if dutch" \
             "enddutch=\endif"
```

那么 ENABLED_SECTIONS 既可有效 english 也可有效 dutch。
也可查看命令\endif, \ifnot, \else, \elseif

\ifnot <section-label>

开始一个条件化文档小节，并使用一个匹配的命令\endif 结束。此条件小节默认为有效，你必须放置该命令的 section-label 到配置标记 ENABLED_SECTIONS 中使得命令\ifnot 无效。

也可查看命令\endif, \if, \else, \elseif

\invariant { description of invariant }

开始一个可描述不变式的段落，此段落将会缩排，段落的文本不会指定一个内部结构，所有视觉增强的命令可以放置到该段落中，多个相邻的\invariant 命令将被组合到一个单独的段落中，每个 invariant 描述都将开启一个新行，另外一个\invariant 命令可能会有若干个不变式，当遇到一个空白行或是其他的小节命令，那么\invariant 将终止。

\note { text }

开始一个可输入提示的段落，此段落将会缩排，段落的文本不会指定一个内部结构，所有视觉增强的命令可以放置到该段落中，多个相邻的\note 命令将被组合到一个单独的段落中，每个 note 描述都将开启一个新行，另外一个\note 命令可能会有若干个提示，当遇到一个空白行或是其他的小节命令，那么\note 将终止。查看\par 命令的例子。

\par [(paragraph title)] { paragraph }

如果一个段落标题已经给定，此命令将开始一个用户自定义的段头。段头将会占用一行，命令后的段落将会缩排。

如果段落标题未给出，此命令将开始一个新段落，也可插入其他段落命令（如\param 或\warning）来终止该命令。

段落的文本不会指定一个内部结构，所有视觉增强的命令可以放置到该段落中，当遇到一个空白行或是其他的小节命令，那么\par 将终止。

例子：

```
/*! \class Test
 * Normal text.
 *
 * \par User defined paragraph:
 * Contents of the paragraph.
 *
 * \par
 * New paragraph under the same heading.
 *
 * \note
 * This note consists of two paragraphs.
 * This is the first paragraph.
 *
 * \par
 * And this is the second paragraph.
 *
 * More normal text.
 */

class Test {};
```

点击此处获得 doxygen 生成 HTML 文档的方法。

\param <parameter-name> { parameter description }

为一个名为<parameter-name>函数参数开始一个参数描述，其次是参数描述。将会检查此参数是否存在，如果在函数声明或定义中，该参数（或者其他参数）的文档丢失或未出现，则将给出一个警告。

\param 命令有一个可选的属性，能指定参数的方向属性，可为“into”和“out”，这有一个 mencpy 函数的例子：

```
/*!
 Copies bytes from a source memory area to a destination memory area,
 where both areas may not overlap.
 @param[out] dest The memory area to copy to.
 @param[in]   src   The memory area to copy from.
 @param[in]   n      The number of bytes to copy
 */
void memcpy(void *dest, const void *src, size_t n);
```

如果参数既可输入也可输出，使用[in，out]来指定方向属性。

段落的文本不会指定一个内部结构，所有视觉增强的命令可以放置到该段落中。多个相邻的\param 命令将被组合到一个单独的段落中，每个参数描述都将开启一个新行，当遇到一个空白行或是其他的小节命令，那么\param 将终止。

\tparam <template-parameter-name> { description }

为一个名为<template-parameter-name>类或函数模板参数开始一个模板参数的描述，其次是模板参数的描述。
等同于\cmdparam

\post { description of the postcondition }

开始一个可描述的后置条件的段落，此段落将会缩排，段落的文本不会指定一个内部结构，所有视觉增强的命令可以放置到该段落中，多个相邻的\post 命令将被组合到一个单独的段落中，每个后置条件描述都将开启一个新行，另外一个\post 命令可能会有若干个后置条件，当遇到一个空白行或是其他的小节命令，那么\post 将终止。

\pre { description of the precondition }

开始一个可描述的前置条件的段落，此段落将会缩排，段落的文本不会指定一个内部结构，所有视觉增强的命令可以放置到该段落中，多个相邻的\pre 命令将被组合到一个单独的段落中，每个前置条件描述都将开启一个新行，另外一个\pre 命令可能会有若干个前置条件，当遇到一个空白行或是其他的小节命令，那么\pre 将终止。

\remarks { remark text }

开始一个可输入一个或多个备注的段落，此段落将会缩排，段落的文本不会指定一个内部结构，所有视觉增强的命令可以放置到该段落中，多个相邻的\remarks 命令将被组合到一个单独的段落中，每个备注都将开启一个新行，另外一个\remarks 命令可能会有若干个备注，当遇到一个空白行或是其他的小节命令，那么\remarks 将终止。

\return { description of the return value }

为一个函数开始一个返回值的描述，段落的文本不会指定一个内部结构，所有视觉增强的命令可以放置到该段落中，多个相邻的\return 命令将被组合到一个单独的段落中，当遇到一个空白行或是其他的小节命令，那么\return 将终止。参看\fn 命令中的例子。

\retval <return value> { description }

为一个名为<return value>函数开始一个返回值的描述，其次是返回值的描述，段落的文本不会指定一个内部结构，所有视觉增强的命令可以放置到该段落中，多个相邻的\retval 命令将被组合到一个单独的段落中，每个返回值描述都将开启一个新行，当遇到一个空白行或是其他的小节命令，那么\retval 将终止。

\sa { references }

为指定的类，函数，方法，变量，文件，URL 的一个或多个交叉引用。开始一个段落，使用“：：”或“#”，作为类和它的一个成员的引用连接符，来组合上述两个名称。在方法名称之后包含一个带括号的参数类型列表，用于若干重载方法或构造器的选择。

等同于命令\see
也可查看“自动链接”章节获取如何创建对象链接的信息。

\see { references }

等价于命令\sa，为了兼容 JavaDoc。

\since { text }

用于指定有效的版本和时间。\since 之后的段落不会指定一个内部结构，所有视觉增强的命令可以放置到该段落中，当遇到一个空白行或是其他的小节命令，那么\since 将终止。

\test { paragraph describing a test case }

开始一个可描述的测试用例的段落，此描述会添加这个测试用例到一个单独的测试列表中，而这两种描述将可交叉引用。在测试列表中的每一个测试用例的前端，都会有一个指明它出处的头。

\throw <exception-object> { exception description }
等同于\exception（查看命令\exception）

注意，与\throws 同义

\todo { paragraph describing what is to be done }
开始一个可描述的 TODO 条目的段落，此描述会添加这个条目到一个单独的 TODO 列表中，而这两种描述将可交叉引用。在 TODO 列表中的每一个条目的前端，都会有一个指明它出处的头。

\version { version number }
开始一个可输入一个或多个版本字串的段落，此段落将会缩排，段落的文本不会指定一个内部结构，所有视觉增强的命令可以放置到该段落中，多个相邻的\version 命令将被组合到一个单独的段落中，每个版本描述都将开启一个新行，另外一个\version 命令可能会有若干个版本字串，当遇到一个空白行或是其他的小节命令，那么\version 将终止。查看命令\author 中的例子。

\warning { warning message }
开始一个可输入一个或多个警告信息的段落，此段落将会缩排，段落的文本不会指定一个内部结构，所有视觉增强的命令可以放置到该段落中，多个相邻的\warning 命令将被组合到一个单独的段落中，每个警告描述都将开启一个新行，另外一个\warning 命令可能会有若干个警告，当遇到一个空白行或是其他的小节命令，那么\warning 将终止。查看命令\author 中的例子。

\xrefitem <key> "(heading)" "(list title)" {text}
此命令是一个类似于\todo，\bug 的集合，用于创建用户自定义文本小节，它会在链接点与关联页面之间自动生成交叉引用，并且能收集关联页面中的所有相同类型的小节。This command is a generalization of commands such as \todo and \bug. It can be used to create user-defined text sections which are automatically cross-referenced between the place of occurrence and a related page, which will be generated. On the related page all sections of the same type will be collected

第一个参数<key>是一个表述小节类型的唯一标识。第二参数是一个加引号的字串，它用于表述第四个参数放置的文本下的小节头，为包含同一标识的所有条目的相关页面，第三个参数（list title）将作为一个标题来使用，一些已预定义的标识：“todo”，“test”，“bug”，“deprecated”。

想想如何使用\xrefitem 命令和它运行后的结果，如果只考虑 todo 列表，它看上去更像是一个 alias（英文输出中）To get an idea on how to use the \xrefitem command and what its effect is, consider the todo list, which (for English output) can be seen an alias for the command
```
\xrefitem todo "Todo" "Todo List"
```

为每个小节都将重复该命令的前三个参数，这使得它非常乏味和易错，、该命令也可用于合并配置文件中的 ALIASES 选项，如定义一个新命令\reminder，可添加以下内容到配置文件中：
```
ALIASES += "reminder=\xrefitem reminders \"Reminder\" \"Reminders\""
```

注意，命令中第二和第三参数使用的转义引号。

--- 创建链接的命令 ---

\addindex (text)
添加文本到 latex 索引中。

\anchor <word>
放置一个不可见的已命名的 anchor（固定点）到文档中，且能使用\ref 命令进行引用。

注意：anchor 目前只能放置在一个标记成页面（使用\page）或主页（使用\mainpage）的注释块中，
也可查看命令\ref

\endlink
终止一个命令\link 开始的链接。
也可查看\link 命令。

\link <link-object>
链接是由 doxygen 自动生成的，且一直有一个指向的对象名作为链接文本。

该命令可用于创建一个对象（一个文件，类或者成员）的链接，并且用户可指定对象的链接文本。此命令可被\endlink 命令终止。所有在\link 和\endling 之间的文本都将作为指向\link 第一个参数<link-object>的链接文本。

查看“自动链接”章节获得更多自动生成链接和有效链接对象的信息。

\ref <name> ["(text)"]
创建一个已命名小节，子小节，页面或 anchor（固定点）的引用，在 HTML 文档中该命令将生成一个指向小节的链接。在一个小节或子小节中将使用小节标题作为链接文本，在 anchor（固定点）中可使用引号间的 text 或者当 text 忽略时使用<name>作为链接文本，而在 latex 文档中，如果<name>引用的是一个 anchor（固定点），该命令将小节生成一个编号或是使用一个页面编号后的文本。

也可查看\page 命令中\ref 的例子。

\subpage <name> ["(text)"]
创建若干页面的一个层次，相同结构也可以使用\defgroup，\ingroup 命令，但在页面中使用\subpage 命令更方便。主页（\mainpage）通常是层次的根节点。

该命令类似于\ref,它用于创建一个到<name>页面标签的引用,并可选择第二个参数指定的 text 作为链接文本。

它不同于\ref 命令之处在于，它只能工作在页面中，在页面中创建一个父子关联，并且子页面使用<name>进行标识。

如果你希望在多个创建的页面中添加它们的结构，可查看\section，\subsection 命令。

注意：每一个页面只能是另一个页面的子页面，而不存在多重关系，因此页面层级必然是一个树型结构。

这有一个例子：

```
/*! \mainpage A simple manual

Some general info.

This manual is divided in the following sections:
- \subpage intro
- \subpage advanced "Advanced usage"
*/

//---------------------------------------------------------

/*! \page intro Introduction
This page introduces the user to the topic.
Now you can proceed to the \ref advanced "advanced section".
*/

//---------------------------------------------------------

/*! \page advanced Advanced Usage
This page is for advanced users.
Make sure you have first read \ref intro "the introduction".
*/
```

\section <section-name> (section title)

创建一个名为<section-name>的小节，小节的标题可使用命令的第二参数指定。

警告：该命令只能工作在相关的页面文档中，而无法工作于其他的文档块中。

\subsection <subsection-name> (subsection title)

创建一个名为<subsection-name>子小节，它的标题可使用命令的第二参数指定。

警告：该命令只能工作在一个相关页面文档块的小节中，而无法工作于其他的文档块中。

也可查看\page 命令中\subsection 的例子。

\subsubsection <subsubsection-name> (subsubsection title)

创建一个名为<subsubsection-name>子小节的小节，它的标题可使用命令的第二参数指定。

警告：该命令只能工作在一个相关页面文档块的子小节的小节中，而无法工作于其他的文档块中。

也可查看\page 命令中\subsubsection 的例子。

\paragraph <paragraph-name> (paragraph title)

创建一个名为<paragraph-name>的段落，它的标题可使用命令的第二参数指定。

警告：该命令只能工作在一个相关页面文档块的子小节的小节中，而无法工作于其他的文档块中。

也可查看\page 命令中\paragraph 的例子。

--- 用于显示例子（将被插入文档中的代码）的命令 ---

\dontinclude <file-name>

用于解析一个源文件，且不论是否被文档完整包含（如同\include 命令所做的），如果你希望将源文件分割成最小的块，并在这些块中间添加文档，那此命令会非常有用。源文件或是目录可使用配置文件中的 EXAMPLE_PATH 标记来指定。

在解析包含\dotinclude 命令的注释块期间，代码段中类和成员的声明和定义将被‘记忆’。

对于单行可使用源文件的单行描述，而显示一行或多行例子可使用\line,\skip,\skipline,\until 命令，为了这些命令将会使用一个内部指针，\dontinclude 可设定这个指针指向例子的第一行。

例子：

```
/*! A test class. */

class Test
{
  public:
    /// a member function
    void example();
};

/*! \page example
 *  \dontinclude example_test.cpp
 *  Our main function starts like this:
 *  \skip main
 *  \until {
 *  First we create a object \c t of the Test class.
 *  \skipline Test
 *  Then we call the example member function
 *  \line example
 *  After that our little test routine ends.
 *  \line }
 */
```

例子文件 example_test.cpp 如下：

```
void main()
{
  Test t;
  t.example();
```

```
}
```
点击此处获得 doxygen 生成 HTML 文档的方法。
也可查看命令\line，\skip，\skipline，\until

\include <file-name>

用于包含一个源文件作为一个代码块，此命令带有一个包含文件名的参数，源文件或是目录可使用配置文件中的 EXAMPLE_PATH 标记来指定。

如果在 EXAMPLE_PATH 标记中指定的<file-name>此例子文件的设定，并非是唯一的，就必须包含<file-name>的绝对路径以消除二义性。

使用\include 命令等价于在文档中插入文件，并使用\code,\endcode 命令限定文件的范围。

\include 命令的主要目的是，避免在包含多个源文件和头文件的例子块中出现代码重复。

源文件的单行描述，可使用组合了\line,\skip,\skipline,\until 命令的\dotinclude 命令。

注意：doxygen 特殊命令是无法在代码块中工作的，但允许在代码块中放置嵌套的 C 风格注释。
也可查看\example，\dontinclude，\verbatim 命令。

\includelineno <file-name>

具备与\include 命令相同的工作模式，但可添加行号到包含文件中。
也可查看\include

\line (pattern)

用于搜索行，除非该命令找到一个非空行否则将搜索至最后用\include 或\dontinclude 包含的例子，如果该行包含指定的模式，将写入输出。

在例子中会使用一个跟踪当前行的内部指针，将找到非空行作为起始行，并设定内部指针。（如果未找到符合要求的行，指针也将指向例子的末端）
也可查看\dontinclude 中的例子。

\skip (pattern)

用于搜索行，除非该命令找到一个包含指定模式的行否则将搜索至最后用\include 或\dontinclude 包含的例子。

在例子中会使用一个跟踪当前行的内部指针，将找到那行作为起始行，并设定内部指针。（如果未找到符合要求的模式，指针也将指向例子的末端）
也可查看\dontinclude 中的例子。

\skipline (pattern)

用于搜索行，除非该命令找到一个包含指定模式的行否则将搜索至最后用\include 或\dontinclude 包含的例子，并将此行写入输出。

在例子中会使用一个跟踪当前行的内部指针，将写入输出的那行作为起始行，并设定内部指针。（如果未找到符合要求的模式，指针也将指向例子的末端）

注意：此命令：

```
\skipline pattern
```

等价于：

```
\skip pattern
\line pattern
```

也可查看\dontinclude 中的例子。

\until (pattern)

将最后包含\include 或\dontinclude 的例子中的所有行写入输出，除非它找到一个包含指定模式的行，而只写入包含指定模式的一行。

在例子中会使用一个跟踪当前行的内部指针，将写入输出的那行作为起始行，并设定内部指针。（如果未找到符合要求的模式，指针也将指向例子的末端）
也可查看\dontinclude 中的例子

\verbinclude <file-name>

在文档中完整包含名为<file-name>文件，此命令等价于在文档中放置\verbatim，\endverbatim 命令限定<file-name>文件。

文件或是目录可使用配置文件中的 EXAMPLE_PATH 标记来指定。

\htmlinclude <file-name>

在 HTML 文档中完整包含名为<file-name>文件，此命令等价于在文档中放置\htmlonly，\endhtmlonly 命令限定<file-name>文件。

文件或是目录可使用配置文件中的 EXAMPLE_PATH 标记来指定。

--- 视觉增强的命令 ---

\a <word>

使用一种指定的字体显示<word>参数，在运行的文本使用此命令建立与成员参数的引用。

例子：

```
  ... the \a x and \a y coordinates are used to ...
```

以下是结果文本：

... the *x* and *y* coordinates are used to ...

\arg { item-description }

除非该命令后遇到一个空白行或是其他\arg 命令，此命令都会有一个参数。它可用于生成一个简单的，无嵌套的参数列表。每一个参数都使用一个\arg 命令开始。

例子：

```
\arg \c AlignLeft left alignment.
\arg \c AlignCenter center alignment.
\arg \c AlignRight right alignment

No other types of alignment are supported.
```

以下是结果文本：

- `AlignLeft` left alignment.
- `AlignCenter` center alignment.
- `AlignRight` right alignment

No other types of alignment are supported.

注意：使用 HTML 命令可创建嵌套列表。
等价于\li

\b <word>

使用一个黑体显示<word>参数，等价于word。也可放置多个字，如multiple words。

\c <word>

使用一个打印字体显示<word>参数，使用该命令引用 Word 的编码，等价于<tt>word</tt>。

例子：

```
... This function returns \c void and not \c int ...
```

以下是结果文本：

... This function returns `void` and not `int` ...

等价于\p，使用打印字体显示多个子，如<tt> multiple words </tt>

\code

开始一个代码块，一个代码块的处理不同于普通文本，它默认将被解析成 C/C++代码，文档化的类和成员的名称将自动被指向文档的链接替代。

也可查看\endcode，\verbatim 命令

\copydoc <link-object>

从指定的<link-object>对象中复制一个文档块，并使用本地命令进行解析。为避免文档块重复的情况，该命令将非常有用，或是用于扩展一个派生成员的文档。

链接对象可指向一个成员（可从属于一个类，文件或是组），一个类，一个名字空间，一个组，一个页面或是一个文件（按顺序检查），注意，如果此对象指向一个成员（从属于函数，变量，类型转换等），为了使其工作，包含它的复合体（类，文件或组）将被文档化。

为复制类成员的文档，可在文档中放置以下内容：

```
/*! @copydoc MyClass::myfunction()
    *   More documentation.
    */
```

如果该成员被重载，可指定相同的参数类型（与成员名之间不留空格），如下：

```
/*! @copydoc MyClass::myfunction(type1,type2) */
```

如果在文档块中查到请求该成员的文本，则需要有能匹配的名称。

Copydoc 命令可用于递归，但递归的每一层级都会暂时中断并记录，如同出现一个错误。but cycles in the copydoc relation will be broken and flagged as an error

注意，简明描述和细节描述都将被复制。查看\cmdcopybrief，\cmdcopydetails 命令，复制注释块中的简明描述或细节描述。

\copybrief <link-object>

\copydoc 命令的简易工作方式，只复制简明描述，而不处理细节描述

\copydetails <link-object>

\copydoc 命令的简易工作方式，只复制细节描述简，而不处理明描述

\dot

开始一个可包含 dot 图形描述的文本段，此文本段可用\enddot 结束。Doxygen 传递文本给 dot，并在输出的结果图片（以及图片映射）中包含这些文本。图中的节点是可点击的，且能链接 URL。可使用 URL 中的\ref 命令能方便地链接 doxygen 中的条目，这有一个例子：

```
/*! class B */
class B {};

/*! class C */
class C {};

/*! \mainpage

  Class relations expressed via an inline dot graph:
  \dot
  digraph example {
        node [shape=record, fontname=Helvetica, fontsize=10];
        b [ label="class B" URL="\ref B"];
        c [ label="class C" URL="\ref C"];
        b -> c [ arrowhead="open", style="dashed" ];
  }
  \enddot
  Note that the classes in the above graph are clickable
  (in the HTML output).
```

```
 */
```

\msc

开始一个可包含消息序列图描述的文本段，查看 http://www.mcternan.me.uk/mscgen/的一些例子。此文本段可用命令\endmsc 结束。

注意：在 msc{…}块中的该文本段只包含消息序列图的一部分。你需要安装 mscgen 工具，才能使用此命令。

这有一个使用\msc 命令的例子

```
/** Sender class. Can be used to send a command to the server.
   The receiver will acknowlegde the command by calling Ack().
   \msc
      Sender,Receiver;
      Sender->Receiver [label="Command()", URL="\ref Receiver::Command()"];
      Sender<-Receiver [label="Ack()", URL="\ref Ack()", ID="1"];
   \endmsc
 */
class Sender
{
   public:
      /** Acknowledgement from server */
      void Ack(bool ok);
};

/** Receiver class. Can be used to receive and execute commands.
   After execution of a command, the receiver will send an acknowledgement
   \msc
      Receiver,Sender;
      Receiver<-Sender [label="Command()", URL="\ref Command()"];
      Receiver->Sender [label="Ack()", URL="\ref Sender::Ack()", ID="1"];
   \endmsc
 */
class Receiver
{
   public:
      /** Executable a command on the server */
      void Command(int commandId);
};
```

\dotfile <file> ["caption"]

在文档中插入一幅 dot 生成的<file>图片。

第一个参数指定了图片的名称。Doxygen 将会在 DOTFILE_DIRS 标记后指定的路径中查找文件名。如果找到将作为 dot 工具的输入文件。结果图片将放置到正确的输入目录中。如果 dot 文件名中包含空格，需要使用引号进行修饰。

第二个参数可选，用于指定图片中显示的标题，即使它没有包含空格也必须放置到引号之前，在标题显示之前引号会被删除。

\e <word>

使用斜体显示<word>，用于突出 word。

等价于\em，为突出多个字可用multiple words

\em <word>

使用斜体显示<word>，用于突出 word。

等价于\e

\endcode

结束一个代码块
也可查看\code 命令

\enddot

结束一个由\dot 开始的块

\endmsc

结束一个由\msc 开始的块

\endhtmlonly

结束一个由\htmlonly 开始的文本块
也可查看\htmlonly 命令

\endlatexonly

结束一个由\latexonly 开始的文本块
也可查看\latexonly 命令

\endmanonly

结束一个由\manonly 开始的文本块
也可查看\manonly 命令

\endverbatim

结束一个由\verbatim 开始的文本块
也可查看\verbatim，\endcode 命令

\endxmlonly

结束一个由\xmlonly 开始的文本块
也可查看\xmlonly 命令

\f$

标记公式文本的开始和结束
也可查看“公式”中的例子

\f[
在单独的一行中间显示加长公式的起始标记。
也可查看“公式”章节和\f]命令

\f]
在单独的一行中间显示加长公式的结束标记。
也可查看“公式”章节和\f[命令

\f{environment}{
在一个指定环境中显示公式的起始标记。

注意：第二参数可选，只用于帮助编辑器（如 Vim），用数量相等的左右大括号定义特定字符生成高亮语法。

\f}
在一个指定环境中显示公式的结束标记。

\htmlonly
开始一个文本块，只能被完整包含在生成的 HTML 文档中。可用 endhtmlonly 命令结束。

此命令可为 doxygen 包含复杂的 HTML 代码（如 applets, java-scripts，html 标记），也可用\latexonly，\endlatexonly 命令提供一份合适的 latex 文档。

注意：在一个只包含 HTML 的块中可已解析环境变量（如$(HOME)）
也可查看\manonly，\latexonly 命令

\image <format> <file> ["caption"] [<sizeindication>=<size>]
在文档中插入一张图片，图片的格式可指定，如果你希望插入多种格式的图片，那么你必须为每种格式设定一次该命令。

第一个参数指定了输出格式，目前只支持 html，latex。

第二个参数指定图片文件名，doxygen 将在 IMAGE_PATH 标记后指定的路径中查找文件名。如果找到将复制到正确的输出目录中。如果图片文件名中包含空格将使用引号修饰。也可以指定一个 URL 来替代文件名，那么 doxygen 将不会复制这个图片，而会检查图片是否存在。

第三个参数可选，用于指定图片中显示的标题。即使它没有包含空格也必须放置到引号之前，在标题显示之前引号会被删除。

第四个参数也可选，用于指定图片的高度或宽度。且只对 latex 输出有效（即 format=latex），sizeindication 既可用于图片的宽度也可用于图片的高度，该尺寸可在 latex 中指定一个有效的尺寸（例如 10cm 或 6 英寸或是一个符号宽度\textwidth）

这有一个注释块的例子：

```
/*! Here is a snapshot of my new application:
    *   \image html application.jpg
```

```
 *   \image latex application.eps "My application" width=10cm
 */
```

以下是一个可查找的与配置文件关联的例子：

```
IMAGE_PATH       = my_image_dir
```

警告：HTML 中所支持的图片格式受限于浏览器的兼容性。Latex 中的图片格式必须是 eps（Encapsulated PostScript）。

Doxygen 无法检查图片格式是否正确，所以你需要自行确认。

\latexonly

开始一个文本块，只能被完整包含在生成的 latex 文档中。可用 endlatexonly 命令结束。

此命令可为 doxygen 包含复杂的 HTML 代码（如图片,公式，特殊字符），也可用\htmlonly，\endhtmlonly 命令提供一份合适的 latex 文档。

注意：在一个只包含 latex 的块中可已解析环境变量（如$(HOME)）
也可查看\manonly，\htmlonly 命令

\manonly

开始一个文本块，只能被完整包含在生成的 MAN 文档中。可用 endmanonly 命令结束。

此命令可在 MAN 页面中直接包含 groff 代码，也可用\htmlonly，\endhtmlonly 命令和\latexonly，\endlatexonly 命令提供一份合适的 HTML 和 latex 文档。

也可查看\latexonly，\htmlonly 命令

\li { item-description }

除非该命令后遇到一个空白行或是其他\li 命令，此命令都会有一个参数。它可用于生成一个简单的，无嵌套的参数列表。每一个参数都使用一个\li 命令开始。

注意：使用 HTML 命令可创建嵌套列表。
等价于\arg

\n

加入一条新行，等价于
,可被打印函数使用。

\p <word>

使用打印字体显示<word>参数，使用该命令可在运行的文本中引用成员函数。
等价于\c

\verbatim

开始一个能被 HTML 和 latex 文档完整包含的文本块，可用\endverbatim 命令结束，在一个完整块中不允许任何注释。

警告：确认每个\verbatim 都对应有一个\endverbatim，否则解析器将出现冲突。
也可查看\code，\verbinclude

\xmlonly
开始一个只能被生成的 XML 完整包含的文本块，可用 endxmlonly 命令结束。

该命令可用于自定义的 XML 标记。
也可查看\htmlonly，\latexonly。

\\
写入一个反斜杠到 HTML 和 latex 的输出中，因为 doxygen 可用此斜杠作为命令之间的分隔符，所以大多数情况下它将被忽略。

\@
写入一个 at 符号（@）到 HTML 和 latex 的输出中，因为 doxygen 可用此符号作为 JavaDoc 命令之间的分隔符，所以大多数情况下它将被忽略。

\~[LanguageId]
使能/取消一个指定的语言过滤器，用于放置不同语言的文档到一个注释块中，使用 OUTPUT_LANGUAGE 标记过滤掉未指定的语言。使用~[LanguageId]可有效一种输出的指定语言，\~则表示在输出中所有语言均有效。（这也是默认设置）

例子：
```
/*! \~english This is english \~dutch Dit is Nederlands \~german Dieses ist
    deutsch. \~ output for all languages.
 */
```

\&
写入一个&符号到输出中，因为此符号在 HTML 中有特殊的含义，所以大多数情况下它将被忽略。

\$
写入一个$符号到输出中，因为此符号可用于扩展环境变量，所以大多数情况下它将被忽略。

\#
写入一个#符号到输出中，因为此符号可用于引用文档化的主体，所以大多数情况下它将被忽略。

\<
写入一个\<符号到输出中，因为此符号在 HTML 中有特殊的含义，所以大多数情况下它将被忽略。

\>
写入一个\>符号到输出中，因为此符号在 HTML 中有特殊的含义，所以大多数情况下它将被忽略。

\%

写入一个\%符号到输出中，因为此符号可防止自动链接到一个文档化类或结构中，所以大多数情况下它将被忽略。

\"

写入一个\"符号到输出中，因为此符号可用于指示一个无格式的文本段，所以大多数情况下它将被忽略。

--- 与 Qt 兼容相关的命令 ---

以下命令是为了兼容 Qt 的类浏览生成器，无需在你的文档使用。

- \annotatedclasslist
- \classhierarchy
- \define
- \functionindex
- \header
- \headerfilelist
- \inherit
- \l
- \postheader

HTML 命令

这有一个可用于插入文档的所有 HTML 命令的列表，注意，虽然这些 HTML 标记为了它的外部格式会被转义成相应的命令，但是一个 HTML 标记的所有属性仍只传递给它的输出（唯一例外是标记中的 HREF 和 NAME 属性）。

<A HREF="...">
一个 HTML 超链接的起始位。（只对 HTML 文件有效）

<A NAME="...">
一个固定点名称的起始位。（只对 HTML 文件有效）

</A>
一个链接或是固定点名称的终止位。（只对 HTML 文件有效）

<B>
使用黑体字显示一块文本的起始位。

</B>
结束<B>的控制。

<BODY> …</BODY>
不产生任何输出的起始位和终止位。

加入一行空白。

<CENTER>
居中文本段的起始位。

</CENTER>
居中文本段的终止位。

<CAPTION>
只在表中使用的一个标题的起始位。

</CAPTION>
只在表中使用的一个标题的终止位。

<CODE>
使用打印字体显示一块文本的起始位。注意，在 C#代码中它等价于\code 命令

</CODE>
结束<CODE>的控制，注意，在 C#代码中它等价于\endcode 命令

<DD>
一个条目描述的起始位。

<DFN>
使用打印字体显示一块文本的起始位。

</DFN>
结束<DFN>的控制。

<DIV>
一种指定风格段的起始位。（只对 HTML 文件有效）

</DIV>
一种指定风格段的终止位。（只对 HTML 文件有效）

<DL>
一个描述列表的起始位。

</DL>
一个描述列表的终止位。

<DT>
一个条目标题的起始位。

</DT>
一个条目标题的终止位。

<EM>
使用斜体字显示一块文本。

</EM>
结束<EM>的控制。

<FORM> …</FORM>
不产生任何输出的起始位和终止位。

<HR>
写入一个水平标尺。

<H1>
一个无编号小节的起始位

</H1>
一个无编号小节的终止位

<H2>
一个无编号子小节的起始位

</H2>
一个无编号子小节的终止位

<H3>
一个无编号子小节中的段的起始位

</H3>
一个无编号子小节中的段的终止位

<I>
使用斜体字显示一块文本的起始位。

<INPUT>
不产生任何输出。

</I>
结束<I>的控制。

<IMG>
只写入属性到 HTML 输出中。

<LI>
一个新条目列表的起始位。

</LI>
一个新条目列表的终止位。

<META>
不生成任何输出。

<MULTICOL>
</MUTLICOL>
将被 doxygen 忽略。

<OL>
一个编码的条目列表的起始位。

</OL>
一个编码的条目列表的终止位。

<P>
一个新段落的起始位。

</P>
一个新段落的终止位。

<PRE>
一个预格式化段落的起始位。

</PRE>
一个预格式化段落的终止位。

<SMALL>
使用一个小号字体显示一块文本的起始位。

</SMALL>
结束<SMALL>的控制。

<SPAN>
一个指定风格的内联文本段落的起始位。（只对 HTML 文件有效）

</SPAN>
一个指定风格的内联文本段落的终止位。（只对 HTML 文件有效）

<STRONG>
一个黑体文本段的起始位。

</STRONG>
一个黑体文本段的终止位。

<SUB>
使用下标显示一块文本的起始位。

</SUB>
结束<SUB>的控制。

<SUP>
使用上标显示一块文本的起始位。

</SUP>
结束<SUP>的控制。

<TABLE>
一个表的起始位

</TABLE>
一个表的终止位

<TD>
一个表的新数据成员的起始位。

</TD>
一个表的新数据成员的终止位。

<TR>
一个表的新行的起始位。

</TR>
一个表的新行的终止位。

<TT>
使用打印字体显示一块文本的起始位。

</TT>
结束<TT>的控制。

<KBD>
使用打印字体显示一块文本的起始位。

</KBD>
结束<KBD>的控制。

<UL>
一个未编码的条目列表的起始位。

</UL>
一个未编码的条目列表的终止位。

<VAR>
使用一个斜体字显示一块文本的起始位。

</VAR>

使用一个斜体字显示一块文本的终止位。

在 doxygen 中确认可用的特殊 HTML 字符：

©
版权符号

&tm;
商标符号

®
注册商标符号

<
小于符号

>
大于符号

&
与号

'
单引号标记

"
双引号标记

‘
左单引号标记

’
右单引号标记

“
左双引号标记

”
右双引号标记

–
 n 破折号（表示数值范围，如 2-8）

—

m 破折号（表示插入符号，如—）

&?uml;
若? 为{A,E,I,O,U,Y,a,e,i,o,u,y}其中之一，写入一个分音给一个字符，如 ä

&?acute
若? 为{A,E,I,O,U,Y,a,e,i,o,u,y}其中之一，写入一个锐音给一个字符，如 á

&?grave
若? 为{A,E,I,O,U,Y,a,e,i,o,u,y}其中之一，写入一个钝音给一个字符，如 à

&?circ
若? 为{A,E,I,O,U,Y,a,e,i,o,u,y}其中之一，写入一个绕舌音给一个字符，如 â

&?tilde
若? 为{A,E,I,O,U,Y,a,e,i,o,u,y}其中之一，写入一个重（chong）音给一个字符，如 ã

ß
写一个清晰的 s（即 ß）到输出中。write a sharp s (i.e. ß) to the output

&?cedil;
若? 为{ c,C }其中之一，写入一个变音符号。

&?ring;
若? 为{ a,A }其中之一，写入一个 a 的长音，如 å

表示一个连续的空格。

最后，在注释块中放置一些不可见的注释 to put invisible comments inside comment blocks，以下将使用 HTML 风格的注释：

```
/*! <!-- This is a comment with a comment block --> Visible text */
```

XML 命令

Doxygen 支持大多数通常用于 C#代码注释的 XML 命令，XML 标记已在定义 C#语言的 ECMA-334（http://www.ecma-international.org/publications/standards/Ecma-334.htm）标准的附录 E 中已经定义了。可惜这份规范太过于笼统，并且给出的一些例子有失水准。

这有一个 doxygen 支持的标记列表：

<c>
标识作为代码块提供的内联文本，等同于使用<tt>text</tt>

<code>
设定源码或输出中的一行或多行，注意，此命令等同于在 C#代码中使用\code ... \endcode，但在其他语言中类似于 HTML 命令<code>...</code>

<description>
作为<list>命令的一部分，用于描述一个条目。

<example>
将一个文本块标记成一个例子，会被 doxygen 忽略。

<exception cref="member">
标示一个方法抛出的异常。

<include>
用于将外部文件导入成一个 XML 块，会被 doxygen 忽略。

<item>
条目列表，只能将其插入到一个<list>的正文中。

<list type="type">
一个列表的起始位，支持 bullet 或 number，以及 table 类型，此列表包含有一定数量的<item>标记，一个 table 类型的列表包含一个头和两个列。

<listheader>
一个 table 类型列表的头的起始位。

<para>

标示一个文本段。

<param name="paramName">
标记名为"paramName"的文本块，等同于使用\param

<paramref name="paramName">
引用一个名为"paramName"的参数，等同于使用\a

<permission>
标示一个成员的安全访问，会被 doxygen 忽略。

<remarks>
标示细节描述。

<returns>
标记一个文档块，它将作为一个函数或方法的返回值，等同于使用\return。

<see cref="member">
引用一个成员，类似于\ref

<seealso cref="member">
在一个“See also”小节中引用“member”的起始位，等同于使用\sa member

<summary>
标示简明描述，类似于\brief

<term>
<list>命令的一部分。

<typeparam name="paramName">
标示一个文本块，它可作为类型参数"paramName"的文档，等同于使用\param

<typeparamref name="paramName">
引用一个名为"paramName"的参数，等同于使用\a

<value>
标示一个属性，会被 doxygen 忽略。

这有一个使用若干上述命令的典型代码块的例子：

```
/// <summary>
/// A search engine.
/// </summary>
class Engine
{
  /// <summary>
  /// The Search method takes a series of parameters to specify the search criterion
```

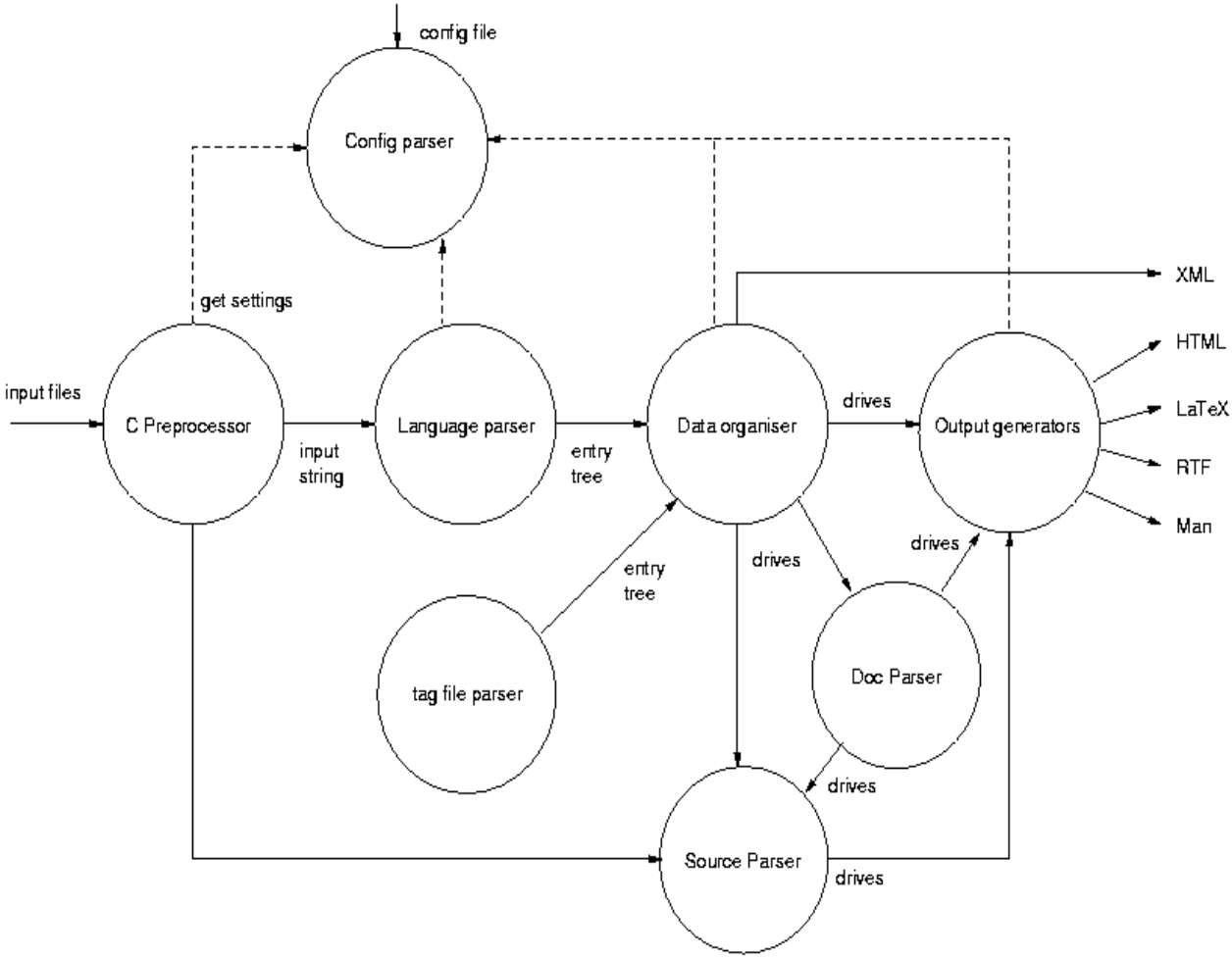
```
  /// and returns a dataset containing the result set.
  /// </summary>
  /// <param name="connectionString">the connection string to connect to the
  /// database holding the content to search</param>
  /// <param name="maxRows">The maximum number of rows to
  /// return in the result set</param>
  /// <param name="searchString">The text that we are searching for</param>
  /// <returns>A DataSet instance containing the matching rows. It contains a maximum
  /// number of rows specified by the maxRows parameter</returns>
  public DataSet Search(string connectionString, int maxRows, int searchString)
  {
    DataSet ds = new DataSet();
    return ds;
  }
}
```

Doxygen's Internals

Doxygen's internals

Note that this section is still under construction!

The following picture shows how source files are processed by doxygen.



Data flow overview

The following sections explain the steps above in more detail.

Config parser

The configuration file that controls the settings of a project is parsed and the settings are stored in the singleton class Config in src/config.h. The parser itself is written using flex and can be found in src/config.l. This parser is also used directly by doxywizard, so it is put in a separate library.

Each configuration option has one of 5 possible types: String, List, Enum, Int, or Bool. The values of these options are available through the global functions Config_getXXX(), where XXX is the type of the option. The argument of these function is a string naming the option as it appears in the configuration file. For instance: Config_getBool("GENERATE_TESTLIST") returns a reference to a boolean value that is TRUE if the test list was enabled in the config file.

The function readConfiguration() in src/doxygen.cpp reads the command line options and then calls the configuration parser.

C Preprocessor

The input files mentioned in the config file are (by default) fed to the C Preprocessor (after being piped through a user defined filter if available).

The way the preprocessor works differs somewhat from a standard C Preprocessor. By default it does not do macro expansion, although it can be configured to expand all macros. Typical usage is to only expand a user specified set of macros. This is to allow macro names to appear in the type of function parameters for instance.

Another difference is that the preprocessor parses, but not actually includes code when it encounters a #include (with the exception of #include found inside { ... } blocks). The reasons behind this deviation from the standard is to prevent feeding multiple definitions of the same functions/classes to doxygen's parser. If all source files would include a common header file for instance, the class and type definitions (and their documentation) would be present in each translation unit.

The preprocessor is written using flex and can be found in src/pre.l. For condition blocks (#if) evaluation of constant expressions is needed. For this a yacc based parser is used, which can be found in src/constexp.y and src/constexp.l.

The preprocessor is invoked for each file using the preprocessFile() function declared in src/pre.h, and will append the preprocessed result to a character buffer. The format of the character buffer is

```
0x06 file name 1
0x06 preprocessed contents of file 1
...
0x06 file name n
0x06 preprocessed contents of file n
```

Language parser

The preprocessed input buffer is fed to the language parser, which is implemented as a big state machine using `flex`. It can be found in the file `src/scanner.l`. There is one parser for all languages (C/C++/Java/IDL). The state variables `insideIDL` and `insideJava` are uses at some places for language specific choices.

The task of the parser is to convert the input buffer into a tree of entries (basically an abstract syntax tree). An entry is defined in `src/entry.h` and is a blob of loosely structured information. The most important field is `section` which specifies the kind of information contained in the entry.

Possible improvements for future versions:

- Use one scanner/parser per language instead of one big scanner.
- Move the first pass parsing of documentation blocks to a separate module.
- Parse defines (these are currently gathered by the preprocessor, and ignored by the language parser).

Data organizer

This step consists of many smaller steps, that build dictionaries of the extracted classes, files, namespaces, variables, functions, packages, pages, and groups. Besides building dictionaries, during this step relations (such as inheritance relations), between the extracted entities are computed.

Each step has a function defined in `src/doxygen.cpp`, which operates on the tree of entries, built during language parsing. Look at the "Gathering information" part of `parseInput()` for details.

The result of this step is a number of dictionaries, which can be found in the Doxygen "namespace" defined in `src/doxygen.h`. Most elements of these dictionaries are derived from the class `Definition`; The class `MemberDef`, for instance, holds all information for a member. An instance of such a class can be part of a file (class `FileDef`), a class (class `ClassDef`), a namespace (class `NamespaceDef`), a group (class `GroupDef`), or a Java package (class `PackageDef`).

Tag file parser

If tag files are specified in the configuration file, these are parsed by a SAX based XML parser, which can be found in `src/tagreader.cpp`. The result of parsing a tag file is the insertion of `Entry` objects in the entry tree. The field `Entry::tagInfo` is used to mark the entry as external, and holds information about the tag file.

Documentation parser

Special comment blocks are stored as strings in the entities that they document. There is a string for the brief description and a string for the detailed description. The documentation parser reads these strings and executes the commands it finds in it (this is the second pass in parsing the documentation). It writes the result directly to the output generators.

The parser is written in C++ and can be found in src/docparser.cpp. The tokens that are eaten by the parser come from src/doctokenizer.l. Code fragments found in the comment blocks are passed on to the source parser.

The main entry point for the documentation parser is `validatingParseDoc()` declared in `src/docparser.h`. For simple texts with special commands `validatingParseText()` is used.

Source parser

If source browsing is enabled or if code fragments are encountered in the documentation, the source parser is invoked.

The code parser tries to cross-reference to source code it parses with documented entities. It also does syntax highlighting of the sources. The output is directly written to the output generators.

The main entry point for the code parser is `parseCode()` declared in `src/code.h`.

Output generators

After data is gathered and cross-referenced, doxygen generates output in various formats. For this it uses the methods provided by the abstract class `OutputGenerator`. In order to generate output for multiple formats at once, the methods of `OutputList` are called instead. This class maintains a list of concrete output generators, where each method called is delegated to all generators in the list.

To allow small deviations in what is written to the output for each concrete output generator, it is possible to temporarily disable certain generators. The OutputList class contains various `disable()` and `enable()` methods for this. The methods `OutputList::pushGeneratorState()` and `OutputList::popGeneratorState()` are used to temporarily save the set of enabled/disabled output generators on a stack.

The XML is generated directly from the gathered data structures. In the future XML will be used as an intermediate language (IL). The output generators will then use this IL as a starting point to generate the specific output formats. The advantage of having an IL is that various independently developed tools written in various languages, could extract information from the XML output. Possible tools could be:

- an interactive source browser
- a class diagram generator
- computing code metrics.

Debugging

Since doxygen uses a lot of `flex` code it is important to understand how `flex` works (for this one should read the man page) and to understand what it is doing when `flex` is parsing some input. Fortunately, when flex is used with the -d option it outputs what rules matched. This makes it quite easy to follow what is going on for a particular input fragment.

To make it easier to toggle debug information for a given flex file I wrote the following perl script, which automatically adds or removes -d from the correct line in the Makefile:

```
#!/usr/local/bin/perl

$file = shift @ARGV;
print "Toggle debugging mode for $file\n";

# add or remove the -d flex flag in the makefile
unless (rename "Makefile.libdoxygen","Makefile.libdoxygen.old") {
   print STDERR "Error: cannot rename Makefile.libdoxygen!\n";
   exit 1;
}
if (open(F,"<Makefile.libdoxygen.old")) {
   unless (open(G,">Makefile.libdoxygen")) {
      print STDERR "Error: opening file Makefile.libdoxygen for writing\n";
      exit 1;
   }
   print "Processing Makefile.libdoxygen...\n";
   while (<F>) {
      if ( s/\(LEX\) -P([a-zA-Z]+)YY -t $file/(LEX) -d -P\1YY -t $file/g ) {
         print "Enabling debug info for $file\n";
      }
      elsif ( s/\(LEX\) -d -P([a-zA-Z]+)YY -t $file/(LEX) -P\1YY -t $file/g ) {
         print "Disabling debug info for $file\n";
      }
      print G "$_";
   }
   close F;
   unlink "Makefile.libdoxygen.old";
}
else {
   print STDERR "Warning file Makefile.libdoxygen.old does not exist!\n";
}

# touch the file
$now = time;
utime $now, $now, $file
```

Perl Module output format documentation

Since version 1.2.18, Doxygen can generate a new output format we have called the "Perl Module output format". It has been designed as an intermediate format that can be used to generate new and customized output without having to modify the Doxygen source. Therefore, its purpose is similar to the XML output format that can be also generated by Doxygen. The XML output format is more standard, but the Perl Module output format is possibly simpler and easier to use.

The Perl Module output format is still experimental at the moment and could be changed in incompatible ways in future versions, although this should not be very probable. It is also lacking some features of other Doxygen backends. However, it can be already used to generate useful output, as shown by the Perl Module-based LaTeX generator.

Please report any bugs or problems you find in the Perl Module backend or the Perl Module-based LaTeX generator to the doxygen-develop mailing list. Suggestions are welcome as well.

Using the Perl Module output format.

When the **GENERATE_PERLMOD** tag is enabled in the Doxyfile, running Doxygen generates a number of files in the **perlmod/** subdirectory of your output directory. These files are the following:

- **DoxyDocs.pm**. This is the Perl module that actually contains the documentation, in the Perl Module format described below.
- **DoxyModel.pm**. This Perl module describes the structure of **DoxyDocs.pm**, independently of the actual documentation. See below for details.
- **doxyrules.make**. This file contains the make rules to build and clean the files that are generated from the Doxyfile. Also contains the paths to those files and other relevant information. This file is intended to be included by your own Makefile.
- **Makefile**. This is a simple Makefile including **doxyrules.make**.

To make use of the documentation stored in DoxyDocs.pm you can use one of the default Perl Module-based generators provided by Doxygen (at the moment this includes the Perl Module-based LaTeX generator, see below) or write your own customized generator. This should not be too hard if you have some knowledge of Perl and it's the main purpose of including the Perl Module backend in Doxygen. See below for details on how to do this.

Using the Perl Module-based LaTeX generator.

The Perl Module-based LaTeX generator is pretty experimental and incomplete at the moment, but you could find it useful nevertheless. It can generate documentation for functions, typedefs and variables

within files and classes and can be customized quite a lot by redefining TeX macros. However, there is still no documentation on how to do this.

Setting the **PERLMOD_LATEX** tag to **YES** in the Doxyfile enables the creation of some additional files in the **perlmod/** subdirectory of your output directory. These files contain the Perl scripts and LaTeX code necessary to generate PDF and DVI output from the Perl Module output, using PDFLaTeX and LaTeX respectively. Rules to automate the use of these files are also added to **doxyrules.make** and the **Makefile**.

The additional generated files are the following:

- **doxylatex.pl**. This Perl script uses DoxyDocs.pm and DoxyModel.pm to generate **doxydocs.tex**, a TeX file containing the documentation in a format that can be accessed by LaTeX code. This file is not directly LaTeXable.
- **doxyformat.tex**. This file contains the LaTeX code that transforms the documentation from doxydocs.tex into LaTeX text suitable to be LaTeX'ed and presented to the user.
- **doxylatex-template.pl**. This Perl script uses DoxyModel.pm to generate **doxytemplate.tex**, a TeX file defining default values for some macros. doxytemplate.tex is included by doxyformat.tex to avoid the need of explicitly defining some macros.
- **doxylatex.tex**. This is a very simple LaTeX document that loads some packages and includes doxyformat.tex and doxydocs.tex. This document is LaTeX'ed to produce the PDF and DVI documentation by the rules added to **doxyrules.make**.

Simple creation of PDF and DVI output using the Perl Module-based LaTeX generator.

To try this you need to have installed LaTeX, PDFLaTeX and the packages used by **doxylatex.tex**.

1. Update your Doxyfile to the latest version using:

```
doxygen -u Doxyfile
```

2. Set both **GENERATE_PERLMOD** and **PERLMOD_LATEX** tags to YES in your Doxyfile.
3. Run Doxygen on your Doxyfile:

```
doxygen Doxyfile
```

4. A **perlmod/** subdirectory should have appeared in your output directory. Enter the **perlmod/** subdirectory and run:

```
make pdf
```

This should generate a **doxylatex.pdf** with the documentation in PDF format.

5. Run:

```
make dvi
```

This should generate a **doxylatex.dvi** with the documentation in DVI format.

Perl Module documentation format.

The Perl Module documentation generated by Doxygen is stored in **DoxyDocs.pm**. This is a very simple Perl module that contains only two statements: an assigment to the variable **$doxydocs** and the customary **1;** statement which usually ends Perl modules. The documentation is stored in the variable **$doxydocs**, which can then be accessed by a Perl script using **DoxyDocs.pm**.

$doxydocs contains a tree-like structure composed of three types of nodes: strings, hashes and lists.

- **Strings**. These are normal Perl strings. They can be of any length can contain any character. Their semantics depends on their location within the tree. This type of node has no children.
- **Hashes**. These are references to anonymous Perl hashes. A hash can have multiple fields, each with a different key. The value of a hash field can be a string, a hash or a list, and its semantics depends on the key of the hash field and the location of the hash within the tree. The values of the hash fields are the children of the node.
- **Lists**. These are references to anonymous Perl lists. A list has an undefined number of elements, which are the children of the node. Each element has the same type (string, hash or list) and the same semantics, depending on the location of the list within the tree.

As you can see, the documentation contained in **$doxydocs** does not present any special impediment to be processed by a simple Perl script.

Data structure describing the Perl Module documentation tree.

You might be interested in processing the documentation contained in **DoxyDocs.pm** without needing to take into account the semantics of each node of the documentation tree. For this purpose, Doxygen generates a **DoxyModel.pm** file which contains a data structure describing the type and children of each node in the documentation tree.

The rest of this section is to be written yet, but in the meantime you can look at the Perl scripts generated by Doxygen (such as **doxylatex.pl** or **doxytemplate-latex.pl**) to get an idea on how to use **DoxyModel.pm**.

Internationalization

Support for multiple languages

Doxygen has built-in support for multiple languages. This means that the text fragments, generated by doxygen, can be produced in languages other than English (the default). The output language is chosen through the configuration file (with default name and known as Doxyfile).

Currently (version 1.7.1), 38 languages are supported (sorted alphabetically): Afrikaans, Arabic, Brazilian Portuguese, Catalan, Chinese, Chinese Traditional, Croatian, Czech, Danish, Dutch, English, Esperanto, Finnish, French, German, Greek, Hungarian, Indonesian, Italian, Japanese (+En), Korean (+En), Lithuanian, Macedonian, Norwegian, Persian, Polish, Portuguese, Romanian, Russian, Serbian, SerbianCyrilic, Slovak, Slovene, Spanish, Swedish, Turkish, Ukrainian, and Vietnamese..

The table of information related to the supported languages follows. It is sorted by language alphabetically. The **Status** column was generated from sources and shows approximately the last version when the translator was updated.

Language	Maintainer	Contact address (replace the at and dot)	Status
Afrikaans	Johan Prinsloo	johan at zippysnoek dot com	1.6.0
Arabic	Moaz Reyad	moazreyad at yahoo dot com	1.4.6
Brazilian Portuguese	Fabio "FJTC" Jun Takada Chino	jun-chino at uol dot com dot br	up-to-date
Catalan	Maximiliano Pin Albert Mora	max dot pin at bitroit dot com [unreachable]	1.6.3
Chinese	Li Daobing Wei Liu	lidaobing at gmail dot com liuwei at asiainfo dot com	1.6.0
Chinese Traditional	Daniel YC Lin Gary Lee	dlin dot tw at gmail dot com garywlee at gmail dot com	1.6.0
Croatian	Boris Bralo	boris dot bralo at gmail dot com	up-to-date
Czech	Petr Přikryl	prikrylp at skil dot cz	up-to-date
Danish	Erik Søe Sørensen	eriksoe+doxygen at daimi dot au dot dk	1.5.4
Dutch	Dimitri van Heesch	dimitri at stack dot nl	up-to-date
English	Dimitri van Heesch	dimitri at stack dot nl	up-to-date
Esperanto	Ander Martinez	dwarfnauko at gmail dot com	1.6.3
Finnish	Antti Laine	antti dot a dot laine at tut dot fi	1.6.0
French	Xavier Outhier	xouthier at yahoo dot fr	1.6.3
German	Jens Seidel	jensseidel at users dot sf dot net	1.6.3

Greek	Paul Gessos	gessos dot paul at yahoo dot gr	up-to-date
Hungarian	Ákos Kiss Földvári György	akiss at users dot sourceforge dot net [unreachable]	1.4.6
Indonesian	Hendy Irawan	ceefour at gauldong dot net	1.4.6
Italian	Alessandro Falappa Ahmed Aldo Faisal	alessandro at falappa dot net aaf23 at cam dot ac dot uk	1.6.0
Japanese	Hiroki Iseri Ryunosuke Satoh Kenji Nagamatsu Iwasa Kazmi	goyoki at gmail dot com sun594 at hotmail dot com naga at joyful dot club dot ne dot jp [unreachable]	1.6.0
JapaneseEn	see the Japanese language		English based
Korean	Kim Taedong SooYoung Jung Richard Kim	fly1004 at gmail dot com jung5000 at gmail dot com [unreachable]	1.6.3
KoreanEn	see the Korean language		English based
Lithuanian	Tomas Simonaitis Mindaugas Radzius Aidas Berukstis -- contact lost --	[unreachable] [unreachable] [unreachable]	1.4.6
Macedonian	Slave Jovanovski	slavejovanovski at yahoo dot com	1.6.0
Norwegian	Lars Erik Jordet	lejordet at gmail dot com	1.4.6
Persian	Ali Nadalizadeh	nadalizadeh at gmail dot com	up-to-date
Polish	Piotr Kaminski Grzegorz Kowal Krzysztof Kral	[unreachable] [unreachable] krzysztof dot kral at gmail dot com	1.6.3
Portuguese	Rui Godinho Lopes	[unreachable]	1.3.3
Romanian	Ionut Dumitrascu Alexandru Iosup	reddumy at yahoo dot com aiosup at yahoo dot com	1.6.0
Russian	Alexandr Chelpanov	cav at cryptopro dot ru	1.6.0
Serbian	Dejan Milosavljevic	[unreachable]	1.6.0
SerbianCyrilic	Nedeljko Stefanovic	stenedjo at yahoo dot com	1.6.0
Slovak	-- searching for the maintainer --		up-to-date
Slovene	Matjaž Ostroveršnik	matjaz dot ostroversnik at ostri dot org	1.4.6

Spanish	Bartomeu Francisco Oltra Thennet David Vaquero	bartomeu at loteria3cornella dot com [unreachable] david at grupoikusnet dot com	up-to-date
Swedish	Mikael Hallin	mikaelhallin at yahoo dot se	1.6.0
Turkish	Emin Ilker Cetinbas	niw3 at yahoo dot com	up-to-date
Ukrainian	-- searching for the maintainer --		1.4.1
Vietnamese	Dang Minh Tuan	tuanvietkey at gmail dot com	1.6.0

Most people on the list have indicated that they were also busy doing other things, so if you want to help to speed things up please let them (or me) know.

If you want to add support for a language that is not yet listed please read the next section.

Adding a new language to doxygen

This short HOWTO explains how to add support for the new language to Doxygen:

Just follow these steps:

1. Tell me for which language you want to add support. If no one else is already working on support for that language, you will be assigned as the maintainer for the language.
2. Create a copy of translator_en.h and name it translator_<your_2_letter_country_code>.h I'll use xx in the rest of this document.
3. Add definition of the symbol for your language in the configure at two places in the script:
    1. After the `f_langs=` is statement, in lower case.
    2. In the string that following `@allowed=` in upper case.

The rerun the configure script such that is generates src/lang_cfg.h. This file should now contain a #define for your language code.

4. Edit language.cpp: Add a

```
#ifdef LANG_xx
#include<translator_xx.h>
#endif
```

Remember to use the same symbol LANG_xx that you added to `lang_cfg.h`. I.e., the `xx` should be capital letters that identify your language. On the other hand, the `xx` inside your `translator_xx.h` should use lower case.

Now, in setTranslator() add

```
#ifdef LANG_xx
    else if (L_EQUAL("your_language_name"))
    {
      theTranslator = new TranslatorYourLanguage;
    }
#endif
```

after the if { ... }. I.e., it must be placed after the code for creating the English translator at the beginning, and before the else {...} part that creates the translator for the default language (English again).

5. Edit libdoxygen.pro.in and add translator_xx.h to the HEADERS line.
6. Edit translator_xx.h:
    - Rename TRANSLATOR_EN_H to TRANSLATOR_XX_H twice (i.e. in the #ifndef and #define preprocessor commands at the beginning of the file).
    - Rename TranslatorEnglish to TranslatorYourLanguage
    - In the member idLanguage() change "english" into the name of your language (use lower case characters only). Depending on the language you may also wish to change the member functions latexLanguageSupportCommand(), idLanguageCharset() and others (you will recognize them when you start the work).
    - Edit all the strings that are returned by the member functions that start with tr. Try to match punctuation and capitals! To enter special characters (with accents) you can:
        - Enter them directly if your keyboard supports that and you are using a Latin-1 font. Doxygen will translate the characters to proper LaTeX and leave the HTML and man output for what it is (which is fine, if idLanguageCharset() is set correctly).
        - Use html codes like ä for an a with an umlaut (i.e. ä). See the HTML specification for the codes.
7. Run configure and make again from the root of the distribution, in order to regenerated the Makefiles.
8. Now you can use OUTPUT_LANGUAGE = your_language_name in the config file to generate output in your language.
9. Send translator_xx.h to me so I can add it to doxygen. Send also your name and e-mail address to be included in the maintainers.txt list.

Maintaining a language

New versions of doxygen may use new translated sentences. In such situation, the Translator class requires implementation of new methods -- its interface changes. Of course, the English sentences need to be translated to the other languages. At least, new methods have to be implemented by the language-related translator class; otherwise, doxygen wouldn't even compile. Waiting until all language maintainers have translated the new sentences and sent the results would not be very practical. The following text describes the usage of translator adapters to solve the problem.

The role of Translator Adapters. Whenever the Translator class interface changes in the new release, the new class TranslatorAdapter_x_y_z is added to the translator_adapter.h file (here x, y, and z are numbers that correspond to the current official version of doxygen). All translators that previously derived from the Translator class now derive from this adapter class.

The TranslatorAdapter_x_y_z class implements the new, required methods. If the new method replaces some similar but obsolete method(s) (e.g. if the number of arguments changed and/or the functionality of the older method was changed or enriched), the TranslatorAdapter_x_y_z class may use the obsolete method to get the result which is as close as possible to the older result in the target language. If it is not possible, the result (the default translation) is obtained using the English translator, which is (by definition) always up-to-date.

For example, when the new trFile() method with parameters (to determine the capitalization of the first letter and the singular/plural form) was introduced to replace the older method trFiles() without arguments, the following code appeared in one of the translator adapter classes:

```
/*! This is the default implementation of the obsolete method
 * used in the documentation of a group before the list of
 * links to documented files.   This is possibly localized.
 */
virtual QCString trFiles()
{ return "Files"; }

/*! This is the localized implementation of newer equivalent
 * using the obsolete method trFiles().
 */
virtual QCString trFile(bool first_capital, bool singular)
{
   if (first_capital && !singular)
      return trFiles();   // possibly localized, obsolete method
   else
      return english.trFile(first_capital, singular);
}
```

The trFiles() is not present in the TranslatorEnglish class, because it was removed as obsolete. However, it was used until now and its call was replaced by

```
trFile(true, false)
```

in the doxygen source files. Probably, many language translators implemented the obsolete method, so it perfectly makes sense to use the same language dependent result in those cases. The TranslatorEnglish does not implement the old method. It derives from the abstract Translator class. On the other hand, the old translator for a different language does not implement the new trFile() method. Because of that it is derived from another base class -- TranslatorAdapter_x_y_z. The TranslatorAdapter_x_y_z class have to implement the new, required trFile() method. However, the translator adapter would not be compiled if the trFiles() method was not implemented. This is the reason for implementing the old method in the translator adapter class (using the same code, that was removed from the TranslatorEnglish).

The simplest way would be to pass the arguments to the English translator and to return its result. Instead, the adapter uses the old `trFiles()` in one special case -- when the new `trFile(true, false)` is called. This is the mostly used case at the time of introducing the new method -- see above. While this may look too complicated, the technique allows the developers of the core sources to change the Translator interface, while the users may not even notice the change. Of course, when the new `trFile()` is used with different arguments, the English result is returned and it will be noticed by non English users. Here the maintainer of the language translator should implement at least that one particular method.

What says the base class of a language translator? If the language translator class inherits from any adapter class the maintenance is needed. In such case, the language translator is not considered up-to-date. On the other hand, if the language translator derives directly from the abstract class `Translator`, the language translator is up-to-date.

The translator adapter classes are chained so that the older translator adapter class uses the one-step-newer translator adapter as the base class. The newer adapter does less *adapting* work than the older one. The oldest adapter class derives (indirectly) from all of the adapter classes. The name of the adapter class is chosen so that its suffix is derived from the previous official version of doxygen that did not need the adapter. This way, one can say approximately, when the language translator class was last updated -- see details below.

The newest translator adapter derives from the abstract `TranslatorAdapterBase` class that derives directly from the abstract `Translator` class. It adds only the private English-translator member for easy implementation of the default translation inside the adapter classes, and it also enforces implementation of one method for noticing the user that the language translation is not up-to-date (because of that some sentences in the generated files may appear in English).

Once the oldest adapter class is not used by any of the language translators, it can be removed from the doxygen project. The maintainers should try to reach the state with the minimal number of translator adapter classes.

To simplify the maintenance of the language translator classes for the supported languages, the `translator.py` Python script was developed (located in `doxygen/doc` directory). It extracts the important information about obsolete and new methods from the source files for each of the languages. The information is stored in the *translator report* ASCII file (translator_report.txt).

If you compiled this documentation from sources and if you have also doxygen sources available the link `doxygen/doc/translator_report.txt` should be valid.

Looking at the base class of the language translator, the script guesses also the status of the translator -- see the last column of the table with languages above. The `translator.py` is called automatically when the doxygen documentation is generated. You can also run the script manualy whenever you feel that it can help you. Of course, you are not forced to use the results of the script. You can find the same information by looking at the adapter class and its base classes.

How should I update my language translator? Firstly, you should be the language maintainer, or you should let him/her know about the changes. The following text was written for the language maintainers as the primary audience.

There are several approaches to be taken when updating your language. If you are not extremely busy, you should always chose the most radical one. When the update takes much more time than you expected, you can always decide use some suitable translator adapter to finish the changes later and still make your translator working.

The most radical way of updating the language translator is to make your translator class derive directly from the abstract class `Translator` and provide translations for the methods that are required to be implemented -- the compiler will tell you if you forgot to implement some of them. If you are in doubt, have a look at the `TranslatorEnglish` class to recognize the purpose of the implemented method. Looking at the previously used adapter class may help you sometimes, but it can also be misleading because the adapter classes do implement also the obsolete methods (see the previous `trFiles()` example).

In other words, the up-to-date language translators do not need the `TranslatorAdapter_x_y_z` classes at all, and you do not need to implement anything else than the methods required by the Translator class (i.e. the pure virtual methods of the `Translator` -- they end with `=0;`).

If everything compiles fine, try to run `translator.py`, and have a look at the translator report (ASCII file) at the `doxygen/doc` directory. Even if your translator is marked as up-to-date, there still may be some remarks related to your souce code. Namely, the obsolete methods--that are not used at all--may be listed in the section for your language. Simply, remove their code (and run the `translator.py` again). Also, you will be informed when you forgot to change the base class of your translator class to some newer adapter class or directly to the Translator class.

If you do not have time to finish all the updates you should still start with *the most radical approach* as described above. You can always change the base class to the translator adapter class that implements all of the not-yet-implemented methods.

If you prefer to update your translator gradually, have a look at `TranslatorEnglish` (the `translator_en.h` file). Inside, you will find the comments like `new since 1.2.4` that separate always a number of methods that were implemented in the stated version. Do implement the group of methods that are placed below the comment that uses the same version numbers as your translator adapter class. (For example, your translator class have to use the `TranslatorAdapter_1_2_4`, if it does not implement the methods below the comment `new since 1.2.4`. When you implement them, your class should use newer translator adapter.

Run the `translator.py` script occasionaly and give it your `xx` identification (from `translator_xx.h`) to create the translator report shorter (also produced faster) -- it will contain only the information related to your translator. Once you reach the state when the base class should be changed to some newer adapter, you will see the note in the translator report.

Warning: Don't forget to compile Doxygen to discover, whether it is compilable. The `translator.py` does not check if everything is correct with respect to the compiler. Because of that, it may lie sometimes about the necessary base class.

The most obsolete language translators would lead to implementation of too complicated adapters. Because of that, doxygen developers may decide to derive such translators from the `TranslatorEnglish` class, which is by definition always up-to-date.

When doing so, all the missing methods will be replaced by the English translation. This means that not-implemented methods will always return the English result. Such translators are marked using word `obsolete`. You should read it **really obsolete**. No guess about the last update can be done.

Often, it is possible to construct better result from the obsolete methods. Because of that, the translator adapter classes should be used if possible. On the other hand, implementation of adapters for really obsolete translators brings too much maintenance and run-time overhead.