Документирование исходного кода для автоматизированного построения программной документации посредством пакета DoxyGen

Руководство

Ревизия: 0.1

Последняя модификация: Vlad Kovtun

Дата последней модификации: 27.09.2009 23:11:00

История изменений

01.05.2006 – Версия 0.1. Первичный документ. Владислав Ковтун

Содержание

Руководство по документированию исходного кода для автоматизированного построения программной документации посредством пакета DoxyGen

1. Конфигурирование DoxyGen.
2. Документирование файла.
3. Документирование класса/структуры.
4. Документирование функции.
5. Документирование перечисления.

1. Конфигурирование DoxyGen

В разделе описываются особенности создания конфигурации для формирования документации по исходному коду проекта.

1.1. Вкладка Project

Вкладка Project содержит описание название проекта документации, версию этого проекта, путь к папке, куда будет размещаться собранный файл документации. Более детально на описании назначения полей данной вкладки остановимся ниже.

На рис. 1 приведено диалоговое окно настройки проекта по формированию программной документации.

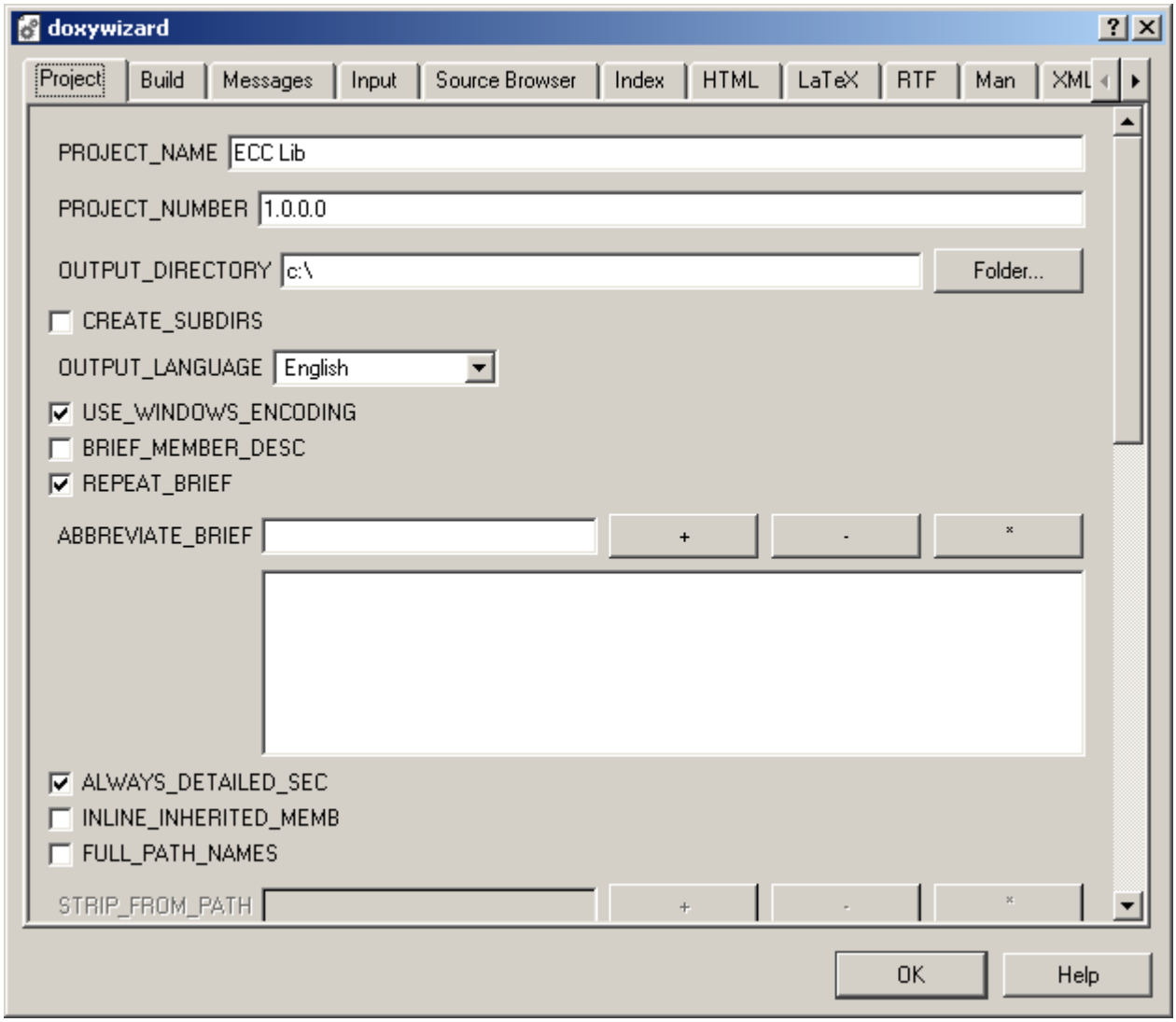

Рис. 1. Диалоговое окно настроек проекта по формированию программной документации

PROJECT_NAME – наименование проекта, для которого формируется документация, будет отображаться на всех страницах документа. Как правило одно слово или несколько слов в кавычках (сказано в документации, но на практике кавычки не являются обязательными).

PROJECT_NUMBER – номер версии проекта, для которого формируется документация, будет отображаться на всех страницах документа.

OUTPUT_DIRECTOTY – путь, где будет располагаться документ. Может быть как абсолютный, так и относительный.

CREATE_SUBDIRS – указывает на необходимость создания отдельной папки для каждого поддерживаемого и генерируемого формата документа.

OUTPUT_LANGUAGE – указывает какой язык используется для формирования документа. Основной язык: **English**, другие поддерживаемые языки: Brazilian, Catalan,

Chinese, Croatian, Czech, Danish, Dutch, Finnish, French, German, Greek, Hungarian, Italian, Japanese, Korean, Lithuanian, Norwegian, Polish, Portuguese, Romanian, **Russian**, Serbian, Slovak, Slovene, Spanish, Swedish и **Ukrainian**.

USE_WINDOWS_ENCODING – указывает о необходимости использования кодировки WINDOWS-1251 (только для Windows платформ).

BRIEF_MEMBER_DESC – позволяет включать в документ комментарии, краткое описание, указанные после членов класса.

REPEAT_BRIEF – указывает на необходимость присоединения комментария, краткого описания, перед детальным описанием функции или члена класса.

ABBREVIATE_BRIEF – позволяет использовать специальные аббревиатуры для формирования краткого описания. Например, при встрече в тексте аббревиатуры \$name, она будет заменена на имя сущности, которую она описывает.

ALWAYS_DETAILED_SEC – позволяет формировать детальное описание, даже если присутствует только краткое описание.

INLINE_INHERITED_MEMB – позволяет в документации отобразить всю иерархию классов, к которой задействован текущий описываемый класс.

FULL_PATH_NAMES – позволяет указывать полный путь в описываемом файле, например в списке используемый файлов.

STRIP_FROM_PATH – позволяет убирать часть пути в описываемом файле.

STRIP_FROM_INC_PATH – позволяет указать путь, по которому следует искать заголовочный файл, содержащий документируемую сущность.

На рис. 2 приведено диалоговое окно настройки проекта по формированию программной документации (продолжение).

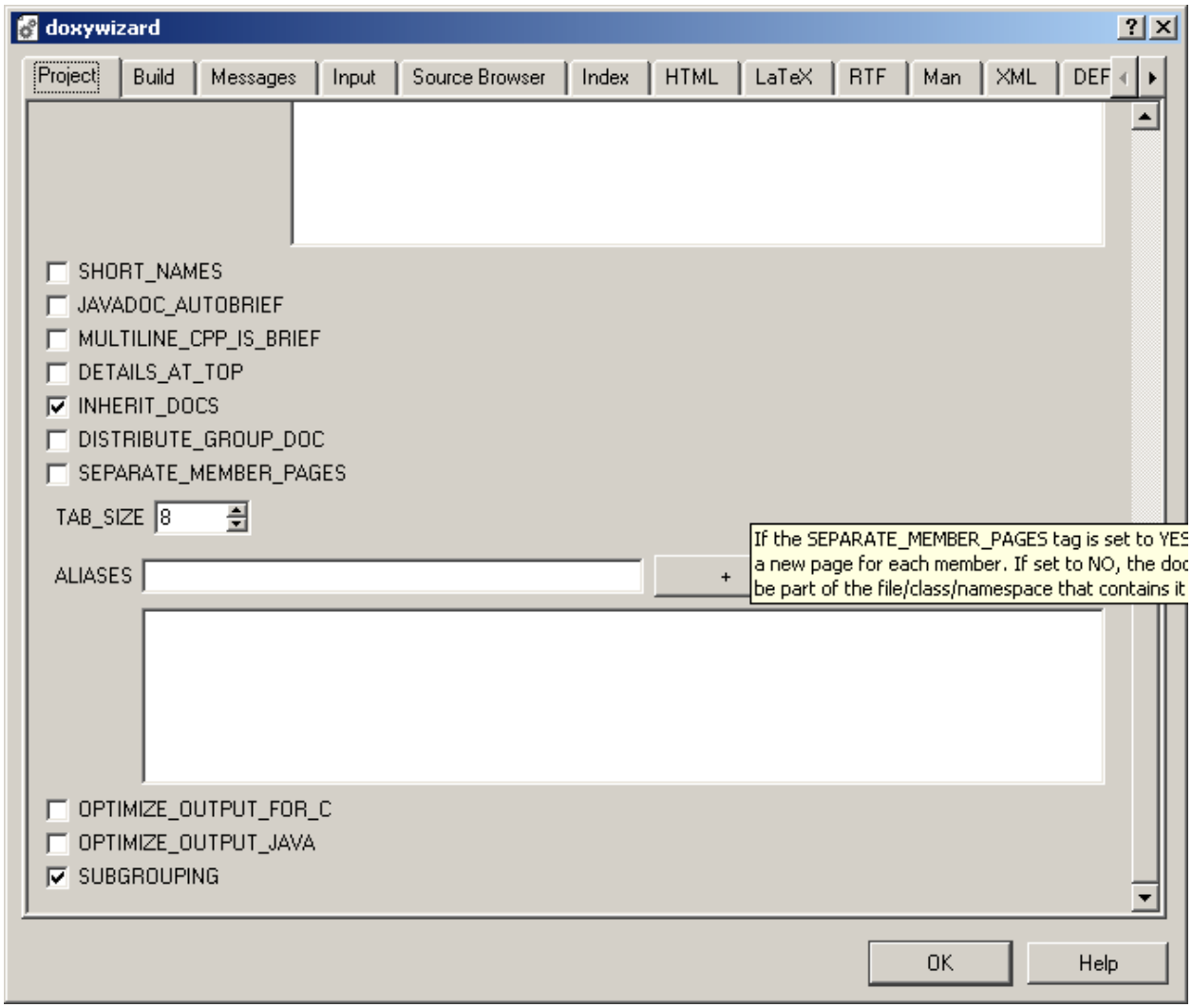

Рис. 2. Диалоговое окно настроек проекта по формированию программной документации (продолжение)

SHORT_NAMES – позволяет формировать файлы документации с короткими именами, что является полезным для файловых систем таких как DOS и др.

JAVADOC_AUTOBRIEF – позволяет формировать документацию, исходя из того, что первая строка до символа '.' в JAVADOC-стиле будет интерпретироваться как краткое описание, в противном случае, первая строка будет интерпретироваться как в Qt-стиле документирования.

MULTILINE_SPP_IS_BRIEF – позволяет распознавать многострочные C++ комментарии //! или /// как краткое описание до тех пор, пока не будет встречена пустая строка.

DETAILS_AT_TOP – позволяет формировать детальное описание сверху как в JAVADOC, в противном случае детальное описание будет приводиться после краткого описания членов.

INHERIT_DOCS – позволяет документировать недокументированные переопределяющие члены из документированных членов иерархии классов.

DISTRIBUTE_GROUP_DOC – позволяет документировать все члены вошедшие в группу, посредством копирования описания первого описанного в группе члена.

SEPARATE_MEMBER_PAGES – позволяет формировать документацию для каждого члена на отдельной странице.

TAB_SIZE – позволяет определить размер табуляции в символах.

ALIASES – позволяет использовать пользовательские типы параграфов. Например **"sideeffect=\par Side Effects:\n"**.

OPTIMIZE_OUTPUT_FOR_C – позволяет формировать документацию наиболее близкую для языка C.

OPTIMIZE_OUTPUT_JAVA – позволяет формировать документацию наиболее близкую для языка Java.

SUBGROUPING – позволяет группировать члены класса, например сгруппировать все public члены.

1.2. Вкладка Build

Вкладка Build содержит информацию об особенности анализа текста исходного кода для формирования файла документации.

На рис. 3 приведено диалоговое окно настройки построения программной документации.

EXTRACT_ALL – позволяет документировать все сущности, имеющиеся в исходном коде, даже если они не содержат комментариев. Private члены и static члены файлов будут сокрыты до тех пор, пока не будет указано EXTRACT_PRIVATE и EXTRACT_STATIC.

EXTRACT_PRIVATE – позволяет включить в документацию информацию о Private членах класса.

EXTRACT_STATIC – позволяет включить в документацию информацию о static членах файлов.

EXTRACT_LOCAL_CLASSES - позволяет включить в документацию информацию локально определенных классах не только в заголовочных, но и в исходных файлах. Для Java смысла не имеет.

EXTRACT_LOCAL_METHODS - позволяет включить в документацию информацию о локальных методах, имеет смысл только для Objective-C.

HIDE_UNDOC_MEMBERS – позволяет скрыть информацию о всех недокументированных членах классов и файлов, в противном случае они будут включены, но без секций документирования.

HIDE_UNDOC_CLASSES – позволяет скрыть недокументированные классы, в противном случае они будут включены, но без секций документирования.

HIDE_FRIEND_COMPOUNDS – позволяет скрыть все дружественные (class|struct|union) объявления, в противном случае они будут включены в документацию.

HIDE_IN_BODY_DOCS – позволяет скрыть все документированные блоки, которые будут найдены внутри тела функций, в противном случае они будут включены в документацию в блок детального описания.

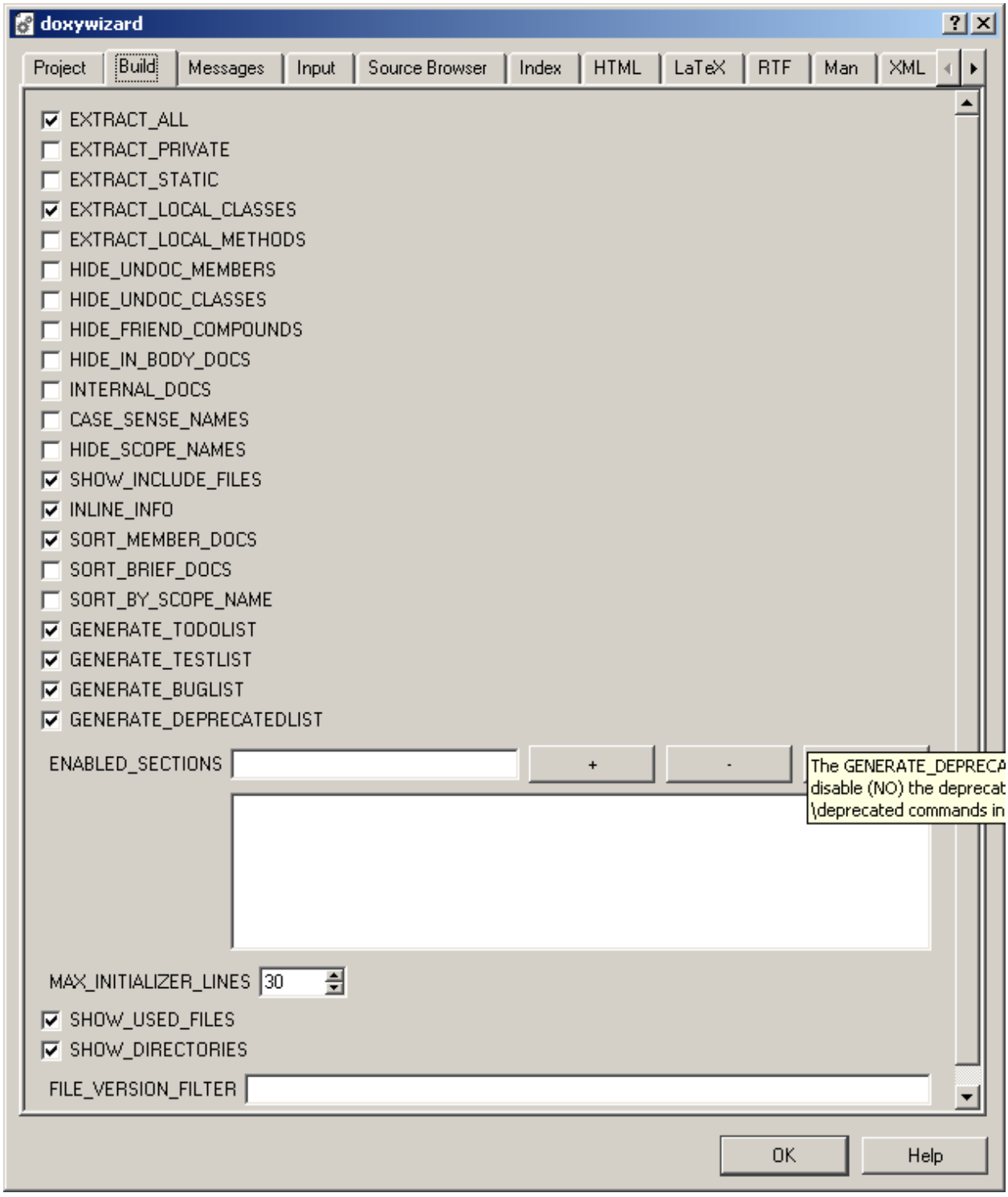

Рис. 3. Диалоговое окно настройки построения программной документации

INTERNAL_DOCS – позволяет включить в документацию все комментарии, которые будут встречены после **\internal** тега, в противном случае они не будут включены в документацию.

HIDE_SCOPE_NAMES – позволяет скрыть во всех именах членов классов и пространств имен часть их имени, что указывает на область видимости, в противном случае они будут указаны в документации.

SHOW_INCLUDE_FILES – позволяет включить в документацию список всех файлов, которые включены в документируемый файл.

INLINE_INFO – позволяет включить тег [inline] в документацию для inline членов.

SORT_MEMBER_DOCS – позволяет сортировать документацию по членам класса и файла в алфавитном порядке, в противном случае они будут приведены в порядке упоминания.

SORT_BRIEF_DOCS – позволяет сортировать краткого описания имен членов файла, пространства имен и класса в алфавитном порядке, в противном случае, они будут приведены в порядке упоминания.

GENERATE_DEPRECATEDLIST – позволяет сформировать список deprecated сущностей из тегов **\deprecated**, в противном случае, он не будет сформирован.

GENERATE_TODOLIST – позволяет сформировать список todo из тегов **\todo**, в противном случае, он не будет сформирован.

GENERATE_TESTLIST – позволяет сформировать test список из тегов **\test** , в противном случае, он не будет сформирован.

GENERATE_BUGLIST – позволяет сформировать bug список из тегов **\bug**, в противном случае, он не будет сформирован.

ENABLED_SECTIONS – позволят использовать теги условного формирования документации, помеченных как **\if** <section-label> ... **\endif** and **\cond** <section-label> ... **\endcond** блоков.

MAX_INITIALIZER_LINES – позволяет определить максимальное число строк, что инициализируют переменную или объявление. Если количество строк превышает указанное, то они будут сокрыты. Используйте 0 для полного сокрытия инициализации. Индивидуальное появление значений переменных и определений может контролироваться посредством использования **\showinitializer** и **\hideinitializer** тега в документации.

SHOW_USED_FILES – позволяет указывать имя файла, где находиться сущность, документация по которой приводится.

SHOW_DIRECTORIES – позволяет указывать иерархи директорий в которой размещены исходные файлы вашего проекта.

1.3. Вкладка Messages

Вкладка Messages содержит информацию о сообщениях, которые формируются в процессе формирования файла документации, на различных его этапах.

На рис. 4 приведено диалоговое окно настроек сообщений, которые формируются в процессе формирования программной документации.

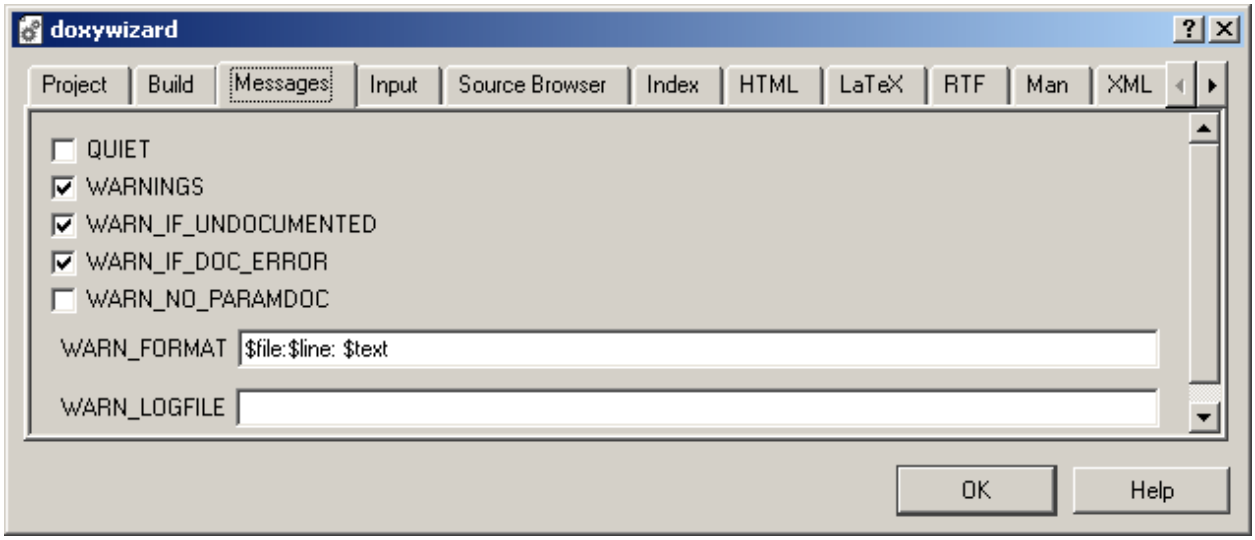

Рис. 4. Диалоговое окно настроек сообщений, которые формируются в процессе формирования программной документации

QUIET – позволяет включить/выключить сообщения, которые генерируются в стандартный output. Возможные значения: YES и NO, где YES значит, что сообщения не генерируются. Если оставлено пустым то значит используется NO.

WARNINGS – позволяет включить/выключить предупреждения которые формируются в стандарный error. Возможные значения: YES и NO, где YES значит, что сообщения не генерируются. Если оставлено пустым то значит используется NO.

WARN_IF_UNDOCUMENTED – позволяет генерировать предупреждения о недокументированных членах, если включен EXTRACT_ALL в положение YES, тогда флаг будет автоматически выключен.

WARN_IF_DOC_ERROR – позволяет формировать предупреждения, в случае выявления потенциальных ошибок в документации, например при появлении недокументированных параметров в документируемой функции или документирование параметров, которые не существуют, или неправильное использование тегов.

WARN_NO_PARAMDOC – позволяет формировать предупреждения для документируемых функций, у которых не документированы их параметрыили возвращаемое значение. Если установлено значение NO (по умолчанию) будет сформировано лишь предупреждение о неправильном или неполном документировании параметров, но не об отсутствии их документации.

WARN_FORMAT – позволяет определить формат сообщения – предупреждения. Строка должна содержать **$file, $line, and $text** теги, которые будут заменены посредством имени файла, номером строки и самим текстом предупреждения.

WARN_LOGFILE – позволяет определить файл лога, который будет содержать все предупреждения и ошибки. Если оставить пустым, то вывод будет осуществляться в stderr.

1.4. Вкладка Input

Вкладка Input содержит информацию о входных данных (исходном), который анализируется и на его основе формируется документация.

На рис. 5 приведено диалоговое окно настроек входных данных, для которых формируется программная документация.

INPUT – позволяет определить файлы и/или директории, что содержать документированные исходные файлы. Можно ввести имя файла, например `myfile.cpp` или директории `/usr/src/myproject`. Отделяются файлы или папки - пробелами. Если оставить пустым, будет использоваться текущая директория.

FILE_PATTERNS – позволяет определить шаблон документированных исходных файлов в директориях. Если оставить пустым, то будут использованы шаблоны: ***.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 – позволяет определить программу или скрипт, который получает последнюю версию каждого файла (обычно из системы управления версиями). Вызов программы осуществляется посредством (через popen()) команд из входного командного файла, где командой является FILE_VERSION_FILTER тег.

Пример использования shell script как фильтра для Unix:

```
FILE_VERSION_FILTER = "/bin/sh versionfilter.sh"
```

Пример shell script для CVS:

```
#!/bin/sh

cvs status $1 | sed -n 's/^[ \]*Working revision:[ \t]*\([0-9][0-9\.]*\).*/\1/p'
```

Пример shell script для Subversion:

```
#!/bin/sh

svn stat -v $1 | sed -n 's/^[ A-Z?\*|!]\{1,15\}/r/;s/ \{1,15\}/\/r/;s/ .*//p'
```

Пример фильтра для for ClearCase:

```
FILE_VERSION_INFO = "cleartool desc -fmt \%Vn"
```

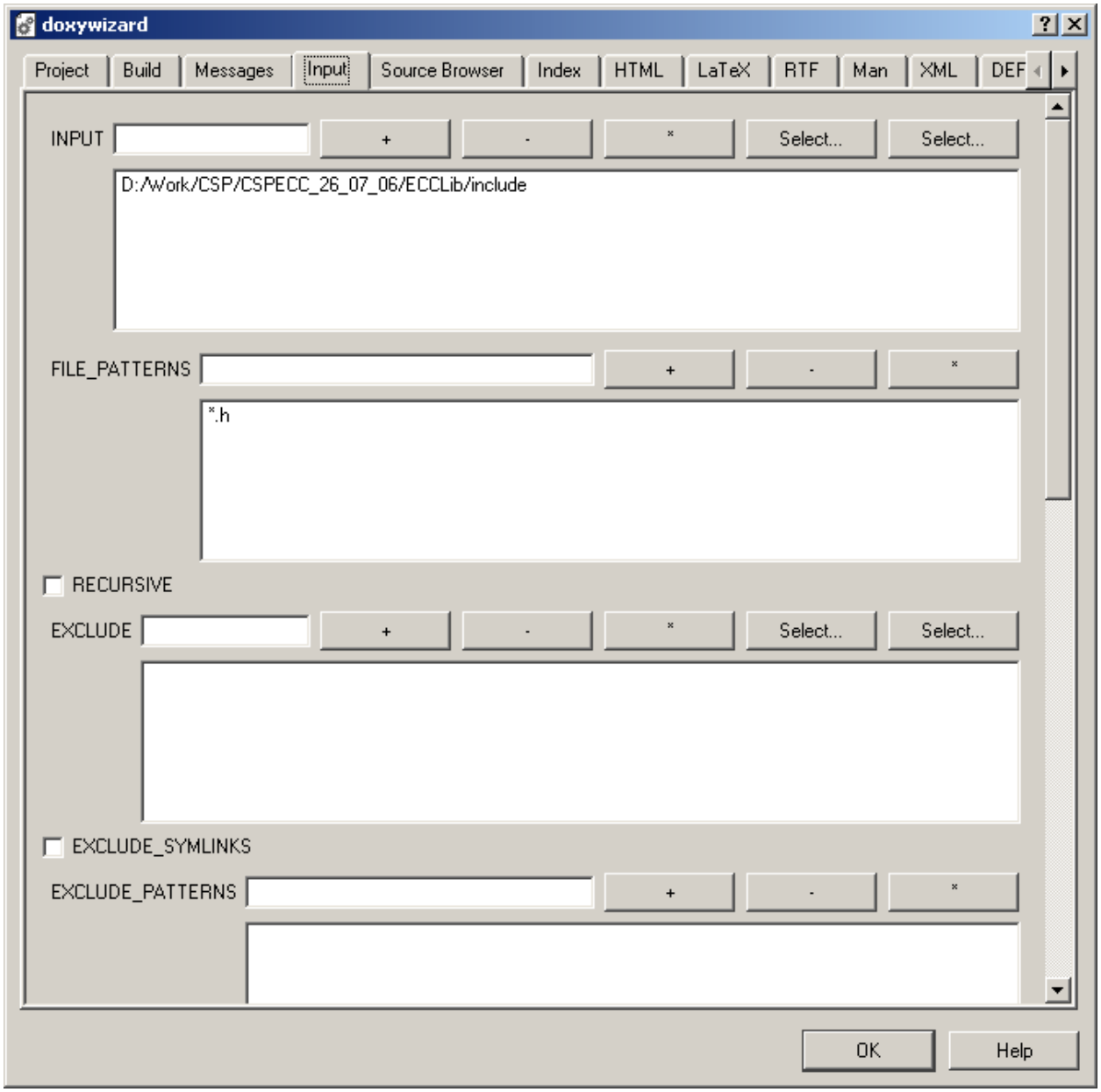

Рис. 5. Диалоговое окно настроек входных данных для которых формируется программная документация

RECURSIVE – позволяет указать следует ли использовать рекурсивный поиск входных файлов. Допустимые значения: YES и NO. Если не указано, то подразумевается NO.

EXCLUDE – позволяет указать файлы и/или директории, которые должны быть исключены из INPUT списка файлов. Этот путь позволяет легко исключить поддиректории из иерархии директорий, чей корень определен в INPUT.

На рис. 6 приведено диалоговое окно настроек входных данных, для которых формируется программная документация (продолжение).

EXCLUDE_SYMLINKS – позволяет выбрать ли или нет файлы или директории, что являются символьными ссылками (особенность файловых систем Unix) исключенными из списка INPUT.

EXCLUDE_PATTERNS – позволяет исключить директории указанные посредством шаблона.

EXAMPLE_PATH – позволяет определить один или несколько файлов или директорий что содержат фрагменты примеров исходного кода, который следует включить в документацию. (следует смотреть тег **\include**).

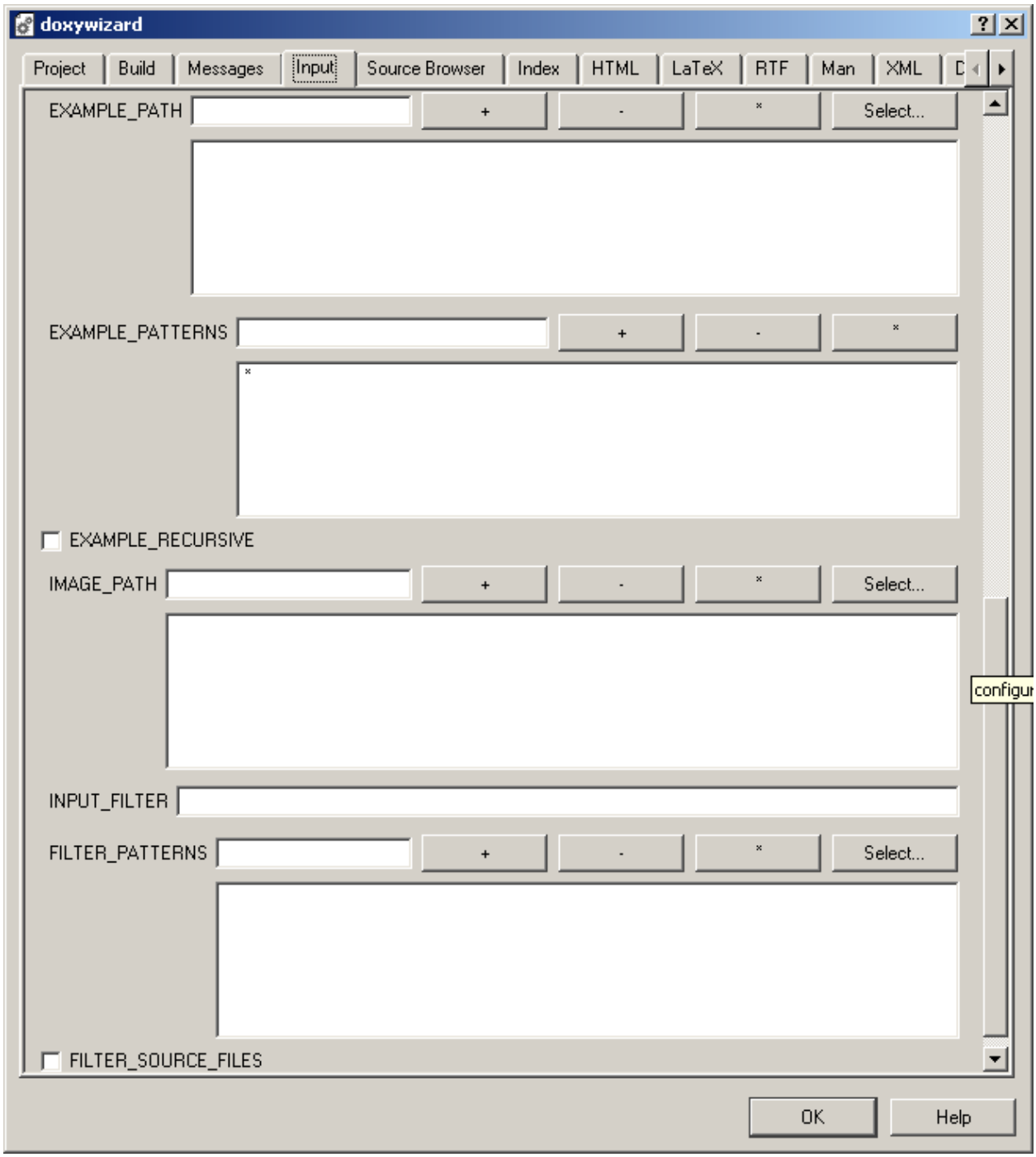

Рис. 6. Диалоговое окно настроек входных данных для которых формируется программная документация (продолжение)

EXAMPLE_RECURSIVE – позволяет рекурсивно искать файлы из INPUT, которые используются с **\include** или **\dontinclude** тегов безотносительно значения **RECURSIVE** тега. Возможные значения: YES и NO. Если оставить пустым, то подразумевается NO.

EXAMPLE_PATTERNS – позволяет указать одну или несколько шаблонов директорий, которые содержат примеры кода, который следует включить в документацию (например ***.cpp** и ***.h**) для фильтрации исходных файлов в директориях. Если оставить пустым – все файлы будут включены.

IMAGE_PATH – позволяет указать один или несколько файлов или директорий, что содержат изображения включенные в документацию (смотрите тег **\image**).

INPUT_FILTER – позволяет определить программу, которая должна вызывать фильтр для каждого входного файла. Программу будет вызвана (посредством `popen()`) командой:

```
<filter> <input-file>
```

где `<filter>` является значение **INPUT_FILTER** тега, и `<input-file>` является имя входного файла. Будет использован output программы-фильтра, которая в него пишет.

FILTER_PATTERNS – позволяет определить фильтр для каждого файлового шаблона. Будет производиться сравнение имени файла с каждым файловым шаблоном и при совпадении будет применяться фильтр. Фильтры являются списком вида: `pattern=filter` (например `*.cpp=my_cpp_filter`). Смотрите **INPUT_FILTER** для более подробной информации об использовании фильтров. Если **FILTER_PATTERNS** является пустым, **INPUT_FILTER** применяется ко всем файлам.

FILTER_SOURCE_FILES – фильтр входных файлов (при условии включения **INPUT_FILTER**) будет использован для фильтрации входных файлов, когда будут просматриваться исходные файлы.

1.5. Вкладка Source Browser

Вкладка Source Browser содержит информацию касательно настроек анализа текста исходного кода.

На рис. 7 приведено диалоговое окно настроек просмотра исходных файлов, для которых формируется программная документация.

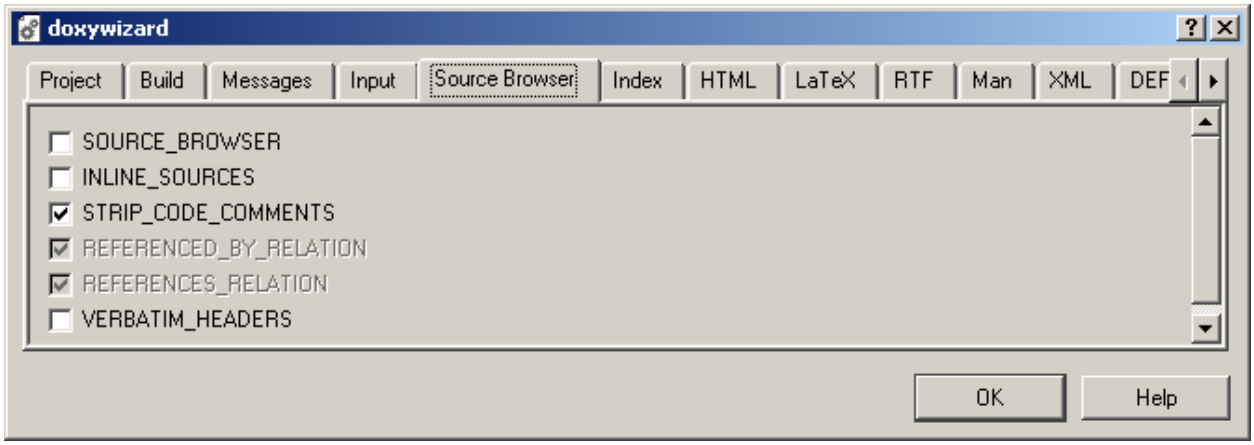

Рис. 7. Диалоговое окно настроек просмотра исходных файлов, для которых формируется программная документация

SOURCE_BROWSER – позволяет сформировать список исходных файлов. Документированные сущности будут обладать кросс-ссылками с этими исходными файлами.

INLINE_SOURCES – позволяет включать тела функций, классов и перечислений непосредственно в документацию.

STRIP_CODE_COMMENTS – позволяет скрывать специальные блоки комментариев кода из генерируемых фрагментов кода. Стандартные C и C++ комментарии всегда будут оставаться видимыми.

REFERENCED_BY_RELATION – позволяет для каждой документированной функции создать список всех документированных функций, которые на нее ссылаются.

REFERENCES_RELATION - позволяет для каждой документированной функции создать список всех документированных сущностей вызывающихся/использующихся, в этой функции.

VERBATIM_HEADERS – позволяет формировать дословную копию заголовочного файла для каждого класса для которых определено включение.

Смотрите секцию: **\class**.

1.6. Вкладка Index

Вкладка Index содержит информацию касательно формирования индекса сущностей, который были выделены в результате анализа исходного кода.

На рис. 8 представлено диалоговое окно настроек посторенние индекса, для формируемой программной документации.

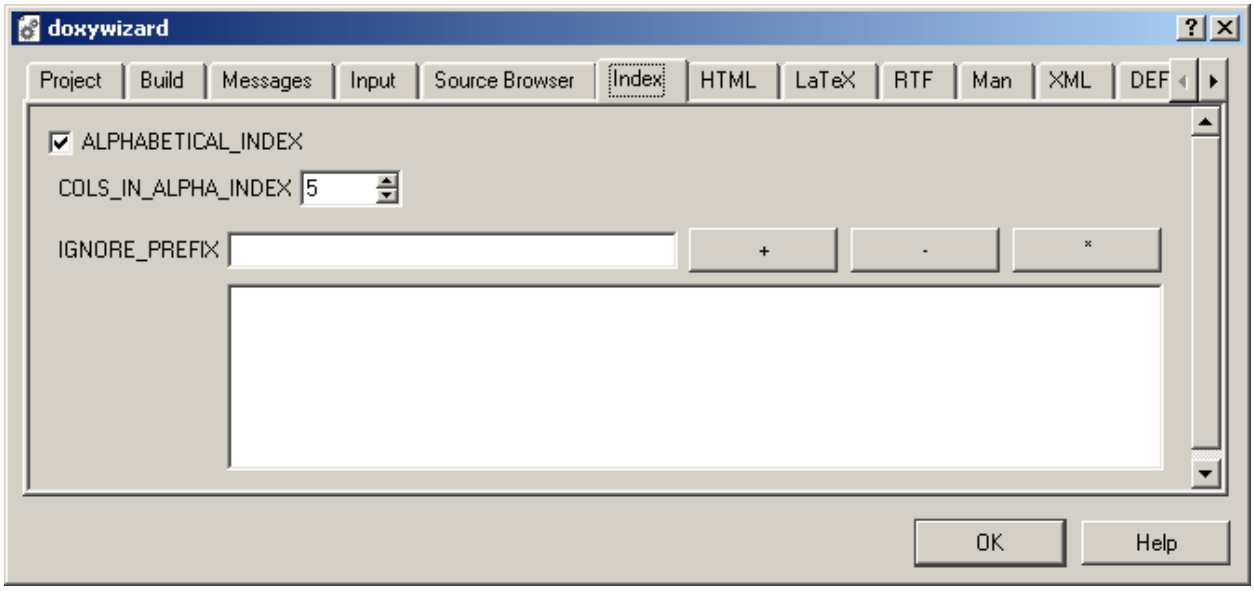

Рис. 8. Диалоговое окно настроек посторенние индекса, для формируемой программной документации

ALPHABETICAL_INDEX – позволяет формировать алфавитный индекс для всех структур. Следует включить, при условии, если проект содержит большое количество классов, структур, объединений и интерфейсов.

COLS_IN_ALPHA_INDEX – позволяет определить (смотрите **ALPHABETICAL_INDEX**) число колонок, на которые этот список будет разделен (может быть число в диапазоне [1..20])

IGNORE_PREFIX – В случае если в проекте все классы начинаются с одинакового префикса, все классы будут расположены в одном заголовочном файле в алфавитном порядке. Этот тег может быть использован для определения префикса (или списка префиксов), что должны быть проигнорированы при формировании индекса заголовочных файлов.

1.7. Вкладка HTML

Вкладка содержит информацию о настройках формирования результирующей документации в формате HTML.

На рис. 9 представлено диалоговое окно настроек посторенние программной документации в HTML формате.

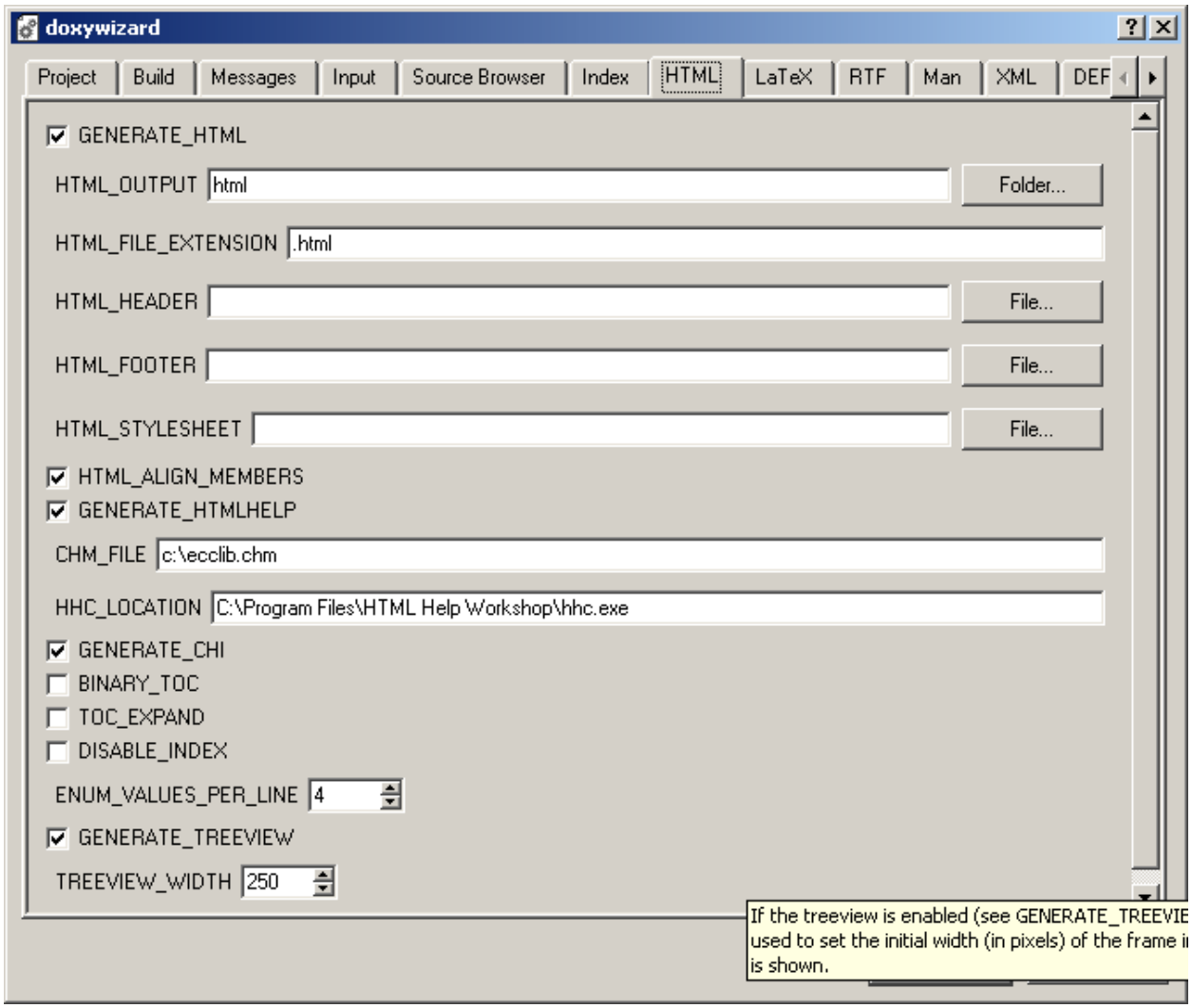

Рис. 9. Диалоговое окно настроек посторенние программной документации в HTML формате

GENERATE_HTML – позволяет формировать документацию в HTML формате.

HTML_OUTPUT – позволяет указать где будет располагаться сформированная HTML документация, в случае указания относительного пути, значение OUTPUT_DIRECTORY будет добавлено в начале относительного пути. Если оставить пустым – будет использоваться путь по умолчанию.

HTML_FILE_EXTENSION – позволяет указать расширения файла, в котором будет храниться документация (например: **.htm, .php, .asp**). Если оставить пустым, то файлы документации будут иметь **.html** расширение.

HTML_HEADER – позволяет определить HTML верхний колонтитул через другой HTML файл для каждой сформированной страницы. Для получения корректного HTML файла, файл с верхним колонтитулом должен содержать как минимум <HTML> и <BODY> теги, также возможно использование страницы стилей. Минимальный пример:

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

Если оставить пустым, то будет использоваться стандартный файл с верхним колонтитулом.

Следующие теги со специальным значением могут использоваться внутри колонтитула: **$title, $datetime, $date, $doxygenversion, $projectname** и **$projectnumber**. Эти теги будут заменены на соответствующие им значения.

Если **CREATE_SUBDIRS** является активированным тег **$relpath$** может быть использован для формирования относительного пути относительно корневого пути для пути по которому будет располагаться HTML документация, т.е. будет использоваться **$relpath$doxygen.css**, для ссылки на стандратную страницу стилей.

HTML_FOOTER – позволяет определить HTML нижний колонтитул для каждой сформированной HTML страницы. Для получения корректного HTML файла, файл нижнего колонтитула должен содержать как минимум `</BODY>` и `</HTML>` тег. Минимальный пример:

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

Если оставить пустым, то будет использован стандартный нижний колонтитул.

Следующие теги со специальным значением могут использоваться внутри колонтитула: **$title, $datetime, $date, $doxygenversion, $projectname** и **$projectnumber**. Эти теги будут заменены на соответствующие им значения.

HTML_STYLESHEET – позволяет определить страницу каскадных стилей, которая будет использована для формирования каждой HTML страницы. Это позволяет для создания необходимого внешнего вида HTML документации. Если оставить пустым, то будет использована стандартная страница стилей.

HTML_ALIGN_MEMBERS – позволяет для членов классов, файлов или пространств имен быть выровненными в HTML посредством таблицы, в противном случае будет использован список.

GENERATE_HTMLHELP – позволяет дополнительно формировать три файла HTML: **index.hhp, index.hhc** и **index.hhk**. Файл **index.hhp** является файлом проекта который может быть прочитан **Microsoft's HTML Help Workshop** для Windows.

HTML Help Workshop содержит компилятор, который позволяет конвертировать всю HTML документацию в один сжатый HTML файл (**.chm**). Сжатый HTML files сейчас использует формат **Windows 98**. Сжатый HTML файл также содержит индекс, таблицу содержимого и позволяет производить поиск по словам d документации. **HTML workshop** также позволяет осуществлять просмотр сжатых HTML файлов.

CHM_FILE – позволяет определить имя результирующего файла **.chm** файла. Возможно указание полного имени файла, в противном случае, он будет размещен в директории с остальными файлами документации.

HHC_LOCATION – позволяет указать абсолютный путь (включая имя фала) **HTML help compiler (hhc.exe)**. Если это поле не пустое, то **HTML help compiler** будет запускаться для сформированного **index.hhp** файла проекта.

GENERATE_CHI – позволяет указать следует ли формировать отдельный **.chi** индексный файл или он должен быть включен в основной **.chm** файл.

BINARY_TOC – позволяет формировать двоичную таблицу содержимого в **.chm** файле, в противном случае – обычную.

TOC_EXPAND – позволяет указать позволяет добавить дополнительные элементы к таблице содержания HTML help документации и к дереву.

DISABLE_INDEX – позволяет получить полное управление над формированием HTML страниц если есть необходимость в отключении индекса и заменой его собственным.

ENUM_VALUES_PER_LINE – позволяет установить количество значений перечисления (в диапазоне [1..20]) что позволяет их группировать в одну линию в сформированной HTML документации.

GENERATE_TREEVIEW – позволяет формировать дерево в боковой части страницы, как индекс (только для HTML Help). Для корректного отображения этой документации требуется браузер поддерживающий JavaScript и frames (например Mozilla 1.0+, Netscape 6.0+ или Internet explorer 5.0+ или Konqueror).

TREEVIEW_WIDTH – позволяет установить ширину окна с деревом, при установленном теге **GENERATE_TREEVIEW**.

2. Документирование

Документирование файла.

```
/// \file file.h
/// \brief Краткое описание.
///
/// Детальное описание файла.
```

При большом количестве разработчиков, работающих над проектом имеет смысл пользоваться также тегом `\author`, `\version` и `\date`.

С целью условного формирования документации используются теги `\if`, `\ifnot`, `\else`, `\endif`, `\elseif`. Их аналогом могут служить `\cond`, `\endcond`.

Рассмотрим пример.

```
/// 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.
```

Использование тега `\remarks` позволяет добавить стандартный дополнительный пункт документации.

3. Документирование класса/структуры

Возможно непосредственное документирование сущности:

```
/// \brief Краткое описание класса.
///
/// Детальное описание класса
Class TestClass
{
public:

}
```

Возможно также отдельное документирование сущности, когда описание может находиться в произвольном месте файла:

```
/// \class TestClass TestClass.h "TestClass.h"
/// \brief Краткое описание класса
///
/// Детальное описание класса.
```

4. Документирование функции

Возможно непосредственное документирование сущности:

```
/// \brief A member function.
/// \param [in] c a character.
/// \param [in] n an integer.
/// \exception std::out_of_range parameter is out of range.
/// \return a character pointer.
const char *member(char,int) throw(std::out_of_range);
```

Возможно также отдельное документирование сущности, когда описание может находиться в произвольном месте файла:

```
/// \fn const char *Test::member(char c,int n)
/// \brief A member function.
/// \param [in] c a character.
/// \param [out] n an integer.
/// \exception std::out_of_range parameter is out of range.
/// \return a character pointer.
```

Возможно, также использование тега `$name` в описании сущности.

5. Документирование перечисления

Рассмотрим перечисление инкапсулированное внутри класса.

```
class Test
{
/// \brief Another enum, with inline docs.
///
/// Detailed description.
enum AnotherEnum
{
V1, /// value 1
V2 /// value 2
};
}
```

Возможно также отдельное документирование сущности, когда описание может находиться в произвольном месте файла:

```
/// \enum Test:::AnotherEnum
/// A brief description of the enum type.
///
/// Detailed description.

/// \var Test:: AnotherEnum AnotherEnum::V1
/// A brief description of the first enum value.
///
/// Detailed description.
```