Introduction

Object Outline generates useful up to date documentation for most C++ projects. This is done by combining the project's source code, the comments in the source code, and any external documentation into a single, coherent meta document. When using Object Outline, external documents will not be lost or forgotten within the source code control system. Instead, all of the external documents will be bound with the source code and/or the subsystem that they document. By intelligently using the source code comments, the documentation does not become a maintenance burden, or worse, forgotten after it is written. Object Outline organizes the classes and functions within a project into hierarchical groups of subsystems, thus avoiding the endless flat list of classes that object browsers present.

By adapting to the comment formatting within your project, Object Outline requires no source code changes and no ugly formatting comment tags. Object Outline can easily be integrated into a daily build procedure and make, this will insure that a project documentation always matches the source code. Documentation is most useful when it is available on-line and as a hard copy. Object Outline can produce documentation as a fully hyper linked HTML file or a fully indexed RTF file that can be printed and bound into a book.

Object Outline currently runs on Windows 95 and Windows NT 3.51, and it works with all of the major C++ compilers for those two platforms. Object Outline also requires a HTML browser and a word processor capable of importing RTF files with tables (Windows 95's wordpad will not work).

How To Order Object Outline

Object Outline can be ordered by calling 1-800-214-4746. Orders can also be faxed to 1-800-657-8141. Object Outline costs $297.00 US dollars per license (+ shipping & handling). The documentation generated by Object Outline can be distributed without limitation. VISA, MasterCard, American Express, and Money orders are all accepted.

If you fax an order please address the fax to Bumble Bee Software and include the following information: your name, address, day time phone number, email address, shipping address, number of copies that you would like to order, credit card type, credit card number, and credit card expiration date.

Sorry, orders outside of the USA and Canada cannot be accepted right now. We are currently working hard to accept international orders. We should be able accept international orders by November.

If you have further questions please feel free to contact us 24 hours a day at 1-800-214-4746 or send us an email.

Please allow one to two weeks for delivery. Thanks!

Tutorial Overview

This tutorial will take about 5 minutes to complete. At the end of the tutorial you will know how to create an Object Outline document and be familiar with the basic functionality of Object Outline.

There is a file called tutorial.cpp in the Object Outline sample directory. This file will be used as the basses for this tutorial. It is a simple program that has three classes, an enumerated type, and several different common commenting styles. Prior to starting the tutorial, open the tutorial.cpp file up with your favorite text editor and briefly study it.

To proceed to the next step in the tutorial start Object Outline then press the next button.

The Options Window

The options' windows is the first window that will appear when Object Outline is started. This window allows the specification of several global options such as: document name, RTF filename, HTML filename, and HTML file generation options.

The first edit box is for the document name. This text will appear at the top of the generated document. This should be a verbose description of the project being documented. Type into the document name field "`My First Document`".

The next two fields allow you to specify the name of the RTF and HTML files. Type "`tut1.rtf`" and "`tut1.html`" in the two fields, respectively. The other options can be left in their default state.

Please press the next button to proceed.

The Pre Processor Window

In this window the include search path and the any #defines needed for parsing the compilers system headers are specified. In the include search path field enter the absolute location for your compilers system headers, and enter any #defines needed by your compiler.

Press here for a list of the commonly needed compiler defines

Press the next button to proceed to the next step in the tutorial.

For more information about the pre processor window, press here.

The Source File Window

This window is used to specify the list of C++ source files that will be incorporated into the document.

To add the tutorial.cpp file

1. Press the add button.
2. Enter the name and path of the tutorial.cpp file that is in the samples sub directory.
3. Press the OK button.

In a normal installation this path would be c:\objectoutline\samples\tutoral.cpp.

Press the next button to proceed to the next step in the tutorial.

The Layout Window

This window is used to organize the generated document into chapters.

To document all of the functions, classes, structs, and enums, defined in tutorial.cpp, please complete the following.

1. First, press the add button.
2. Next change the entry type to ‘‘Source File’’ and enter the name and path of the tutorial.cpp file.
3. Lastly press the OK button.

To insert an external ASCII document into the master document, do the following.

1. Press the add button.
2. Change the entry type to ‘‘Document Inserting’’ and enter the name and path of the tutorial.txt file.
3. Press the OK button.

In a similar manner, chapters, sub chapters, classes, functions, and non ASCII external documentation can be added to the generated document.

Press the next button to proceed to the next step in the tutorial.

For more information about the layout window, press here.

The Comment Filter Window

This window is used to filter the source code comments. Many common commenting styles have extra information that is not appropriate, or is redundant, with the information generated by Object Outline. For example, listing the function name in the comment has no value in the generated document because the function name is parsed directly from the source code. This window can be used to remove this and other types of unneeded information from the comment.

To tell Object Outline to remove all occurrences the word “description”:

1. Press the add button.
2. Enter “description” as the tag name.
3. Check the “remove always” and “remove text before” check boxes.
1. Press OK.

This will tell Object Outline to remove all occurrences of the word description from the comments and remove all of the text in the comments before the word “description”. This will remove the redundant function names from the comments in tutorial.cpp file.

Press the next button to proceed to the next step in the tutorial.

For more information about the comment window, press here.

Documentation Generation

Once the document is configured then it is time to generate a document.

To generate a document, first select File | Generate from the main menu. This will bring up the generate dialog, next, press the start button. When Object Outline has finished processing the source files, press the view RTF or view HTML buttons to display the generated document.

Note: You must have a HTML browser installed on your system to view the HTML document, and a word processor that supports importing RTF files with tables. Windows 95's WordPad ignores many of the advanced formatting codes and should not be used.

Using the C++ Preprocessor

Object Outline follows the Microsoft Visual C/C++ include file search convention. As quoted from the Microsoft VC++ manual.

> Angle-bracket form: This form causes the preprocessor to search for include files first along the path specified in the preprocessor window, then along the path specified by the INCLUDE environment variable.
>
> The preprocessor stops searching as soon as it finds a file with the given name. If you specify a complete, unambiguous path specification for the include file between two sets of double quotation marks (**" "**), the preprocessor searches only that path specification and ignores the standard directories.
>
> If the filename enclosed in double quotation marks is an incomplete path specification, the preprocessor first searches the "parent" file's directory. A parent file is the file containing the **#include** directive. For example, if you include a file named `file2` within a file named `file1`, then `file1` is the parent file.
>
> Include files can be "nested"; that is, a **#include** directive can appear in a file named by another **#include** directive. For example, `file2`, above, could include `file3`. In this case, `file1` would still be the parent of `file2` but would be the "grandparent" of `file3`.
>
> When include files are nested, directory searching begins with the directories of the parent file and then proceeds through the directories of any grandparent files. Thus, searching begins relative to the directory containing the source currently being processed. If the file is not found, the search moves to directories specified in the include search path. Finally, the directories specified by the INCLUDE environment variable are searched.

Before using Object Outline the #defines needed for the operating system and compiler must be defined. If the defines are not correct Object Outline will not parse the system and compiler header files correctly. The documentation for each compiler should be referenced for this information if your compiler or project type is not listed in predefined configuration list. Most parsing problems can be traced back to a missing #define.

Metrics

Object Outline can automatically generate a variety of object oriented metrics. Each metric is calculated for the entire system and for each chapter. When calculating metrics for the chapters, the values are compared to the system wide average. The metrics are:

Total number of classes - This is the total number of classes in the current chapter including all subchapters. This can be used as a first order size estimation, and to insure that subsystems are not too large.

Total number of statements - This metric is a measure of the size of the system. A statement is any line inside a function body that ends in a semicolon, an if statement, or a while statement.

Total number of free standing functions - This is the total number of functions that are not part of a class. Too many free standing functions in an object oriented system may be an indicator of design problems. The percentage of free standing functions assigned to the current chapter is also calculated and documented. If the system being document does not aggressively use classes, this metric can be used as a indicator of the projects progress.

Average number of public member functions per class - This is the average number of public member functions. Too many public member functions can be a indicator that the object is doing too many things. Classes with more than twenty public member functions that are not user interface classes should be looked at.

Average number of protected member functions per class - This is the average number of protected member functions.

Average number of parameters per function - This is the average number of parameters per function. Too many parameters in an object oriented system may be a sign of design problems. Typical averages for object oriented systems is less than one.

Average number of statements per function - This is the average number of statements per functions. This metric includes both free standing and the member functions. A high number in an object oriented system (larger than 10) may indicate the objects are to large and the problem is not composed at a fine enough level. The current chapter average is compared to the systems average.

Percent of functions documented - This is a the percentage of functions that have documentation. A low percentage indicates that the system is not commented enough. Acceptable threshold levels vary from project to project.

Comment Filtering Example

The comment filter window is the key to using Object Outline without changing your source code. This window is used to configure the build in comment filter. The filter can remove redundant information (like the function names) from the comments, and can be used to format the comments in the finial document.

The filter divides the comments into sections delimited by user defined tags. Sections in the comments can be filtered differently depending upon what tag is before or after it. For example, this is a comment from the C runtime library of a popular C++ compiler vendor:

```
/***
*char *strstr(string1, string2) - search for string2 in string1
*
*Purpose:
*       finds the first occurrence of string2 in string1
*
*Entry:
*       char *string1 - string to search in
*       char *string2 - string to search for
*
*Exit:
*       returns a pointer to the first occurrence of string2 in
*       string1, or NULL if string2 does not occur in string1
*
*Uses:
*
*Exceptions:
*
*******************************************************************************/
```

First, Object Outline automatically removes the asterisks from the comment. The underlined words are the tags defined for this commenting style. Note, that the comment is made of six standard sections, the sections are delimited with standard tags. The first tag is Purpose:, the text before this tag is redundant because Object Outline will get this information from the source code, therefor Object Outline should be configured to remove this tag from the comment. Select the "Always remove tag from comments" and select the "remove text before tag" check box in the edit tag window. The Entry and Exit tags may or may not need removing depending upon your style preferences. The Uses and Exceptions tags should only be removed if they have no text after them. To setup this behavior, select "Remove tag only when empty" and "remove text after tag". After the filter the comment would look like this.

finds the first occurrence of string2 in string1
*char *string1 - string to search in*
*char *string2 - string to search for*

returns a pointer to the first occurrence of string2 in string1, or NULL if string2 does not occur in string1

Also, the tag can be specified with standard wildcards. Some experimentation may be needed to configure the filter correctly.

Document Layout Overview

All but the smallest systems are impossible to understand without some level of grouping above the class level. Object Outline lets you divide your projects documentation into subsystems. This avoids the common problem with many automatic documentation systems of the endless flat list of classes and functions organized alphabetically.

This window is used to describe the makeup and relationships between subsystems in the software being documented. This window allows you to divide the system into a hierarchy of nested subsystems. The subsystems can be nested up to three levels deep, allowing the document to scale to large systems.

This window allows you to assign all of the classes, structs, functions, and enums defined in a specified file to a single subsystem. Also,the subsystem can be described at the file and class level. Each subsystem also have external design documents assigned into it. If the external documents are text files, then they can be imported directly into the document. However, if the external documents are not text files then they can have a hyper link reference added to them from the generated document.

Drag and drop can be used to easily maintain.

Integrating Object Outline Into Make.

Object Outline comes with two versions. An interactive version that allows the point and click editing, creating, and processing of the configuration files. Also, a command line program that is designed to be placed into a makefile and be run from the command line.

The command line version is called objout.exe. It has the following command line options.

objout [-IXDo] [-nohtml] [-nortf] -ool "config file name"

-I	Add the following path to the list of include directories.
-X	Ignore environment variable.
-D	define the macro.
-nohtml	Do not a create a html file.
-nortf	Do not a create a rtf file.
-o	Place all output files into the following directory.
-ool	Use this configuration file. This is required option.

Object Outline does not accept response files.

Getting In Touch With Us

If the information needed to solve your problem is not available in the online documentation:

Please feel free to contact Bumble Bee Software directly at support@bbeesoft.com. We kindly request that you supply enough information to reproduce the problem.

All feedback is welcomed, including ideas to improve the product, bug reports, documentation problems, and feedback about the WEB site. We want to here what you think! Please feel free to drop us a note at support@bbeesoft.com. Also, don't forget about the Bumble Bee Software WEB site at www.bbeesoft.com.

End User License Agreement

Object Outline, is owned by Bumble Bee Software. It is protected by U.S. copyright laws and other laws by international treaties.

Bumble Bee Software grants the use of Object Outline on a single computer by a single user. Object Outline may not be installed on a network unless separate copies are purchased for each user that will access it. The documentation generated by Object Outline can be distributed without limitations.

In no event will Bumble Bee Software be liable for any indirect, special, incidental or consequential damages (including loss of profit) where based on contract, tort, or any other legal theory, even if Bumble Bee Software was advised of the possibility of such damages. Because some states do not allow the exclusion of limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

Bumble Bee Software warranties that the unaltered software will substantially perform the function described in the users manual of sixty (60) days after purchase. Bumble Bee Software sole and exclusive obligation, and your sole exclusive remedy, under this warranty shall be limited to Bumble Bee Software using reasonable efforts to correct material, documentation, reproducible defects in the unaltered software that you describe and document to Bumble Bee Software during the 60 day warranty period. In the event that Bumble Bee Software cannot correct the problem within a reasonable period, Bumble Bee Software, at Bumble Bee Software's discretion, will replace the defective software or refund to you the amount that you paid Bumble Bee Software for the defective software and cancel this agreement and the licenses granted herein. In such event, you agree, to return to Bumble Bee Software all copies of the software and its documentation.

Object Outline Top Level Description

Object Outline is configured in five steps.

The first window is the options window. In the options window you enter the RTF file name, HTML file names, and configure various options that affect the entire generated document.

The second window is the C++preprocessor window. This window is used to tell Object Outline where the various header files are found, and what macros to define before processing.

The third window is the source file window. The source file window is used to specify the source files to document.

The forth window is the layout window. The layout window is used to setup the structure and organization of the generated document. It is also used to specify how the external documentation should be integrated with the generated document.

The last window is the comment filter configuration window. This window specifies how the comments in the source code should be filtered before they are inserted into the generated document.

The Options Window

This window is used to configure miscellaneous aspects of the generated document such as:

Document Name - The text entered into this edit control appears on the cover page of the generated document. This can also be changed by editing the name of the root item in the layout window. In most cases this is the name of the product that you are documenting.

RTF output file name - This is the name of the generated RTF file. The extension .RTF will be automatically added if an extension is not given. Leave this field blank if you do not want an RTF file generated.

HTML output file name - This is the name of the base HTML file. The extension .HTML will be added automatically if an extension is not given. Leave this field blank of you do not want an HTML file generated.

Document protected members - This option, when checked, will document all protected member functions, classes, data, and enums. When turned off, Object Outline only documents the public part of all classes. Object Outline does not document the private sections of the class. This options affects the HTML and RTF files.

Generate metrics - This option, checked, will cause Object Outline to generate and insert project metrics for each subsystem and for the entire project. Metrics may need to be turned off when generating documentation for audiences other than the projects authors.

Show Functions in Table of Contents - This option when checked, will list all of the functions in the table of contents. If this option is turned off just a single link is added to the first function in each chapter. This option should be turned on for C code or for code that is not object oriented.

The HTML only options are:

> Force output into one file - This option overrides the default behavior of putting each chapter in its own HTML file. This options should only be used if the target system does not support long file name, or if the system documentation is small enough that it can fit into one file. This option is mutually exclusive with the split view option. For large systems, this should be left off. This option is only valid for HTML files.
>
> Generate Split View - This option when checked generates a HTML file set with two frames. The frame on the left contains the system wide table of contents and the right frame contains the currently selected item. This allows the user to keep the table of contents always visible while navigating though the document. This option is only valid for HTML files.
>
> Add Links in function argument - This option, when set on, forces the function argument to be hyper linked when a known class is used as a parameter. This option is only valid for HTML files.
>
> Add Links in document text - This option, when set on, will cause Object Outline to search the comment text and automatically add hyper links to classes that appear in other parts of the document. This option is only valid for HTML files.
>
> Add Links to include File - This option, when set to off will not generate a link back to the parsed #include files. This should be used if the source code is not or should not be made available with the HTML document. For example, you may want to put Object Outlines documents on the Internet, but not your header files. This option is only valid for HTML files.

Background graphic - This option allows the user to specify a background graphic for all of the HTML files generated. The graphic can be the company logo, a warning that the information in the page is private and confidential, or just a mono color graphic. The file must be either an BMP and GIF file.

For more information about Object Outline metrics, press here.

The C++ Pre Processor Window

This window is used to configure the Object Outline C++ preprocessor. The preprocessor is responsible for processing #include files, macros, and condition compilations. This window is needed to correctly parse compiler and OS headers files. An incorrectly configured preprocessor window can cause the Object Outline parser to fail, and the generated documentation to wrong.

Re-Parse headers for each module - This option disables Object Outlines precompiled header feature. This options should only be turned on if the source code heavily uses the preprocessor and the order that header files are included significantly affects the generated code. For large documents, Turning this option on will slow the document generation down by an order of magnitude.

Compile and Project Type - This option allows the use of several common configurations and compilers. If you compiler or project type is not listed in this control, you will need to add all of the necessary defines in the Additional defines control.

Additional defines - This edit box is used to define macros that should be defined across the life time of the source code parsing. The standard defines for your compiler defines should be placed in this window. The macros should be separated with commas, and the values should be separated with equals signs.

For example: NDEBUG,_WIN32,_MSC_VER=1000,_M_IX86=400,_WINDOWS

Ignore the INCLUDE environment variable - This option, when enabled, ignores the path specified by the INCLUDE environment variable.

For more information about the pre processor window, press here.

The Source Window

This window is used to tell Object Outline where to find the source code for the current project being documented.

Object Outline supports recursive directory searching with wildcards. If all of the source code for the project is under a single directory tree, then the simplest configuration is to point Object Outline at the root of the source code tree with a recursive directory search of *.cpp. This will allow the addition, removal, and renaming of source files without updating the project file. However, if only a simple document is needed describing a couple of files or if files are in just a single sub directory, then they can be individually specified. Typically only the source files are listed in this window.

The Source Edit Window

This window is used to add a source file, or a group of source files to the document. A group of files can be added in one line using a wildcard, for example "*.cpp", or individual files can be specified. If the include sub folders option is specified, then Object Outline will search in the specified subdirectory and all of its children. If all of the source code for the project being document is in a single source tree, then the entire tree can be specified in just one line. Typically the .CPP files are listed in this window.

The Layout Window

The layout windows is where the overall structure and layout of the generated document is described. This layout window is used to divide the generated documentation into chapters. It is also used to assign what classes and functions go with what chapters. Chapters can be nested up to three levels deap. The contents of the chapters (the stuff documented in them) can be: classes, functions, entire header files, metrics, and external documents. The external documents will usually document the chapters that they are inserted into. Chapters and the contents of chapters can be easily rearranged by using drag and drop.

This window allows you to add, edit, and delete items from the layout.

For more information about the layout window, press here.

The Layout Edit Window

The layout of the generated document is made up of the following items:

Chapter - This layout item is a container of other layout items, including other chapters. Chapters can be nested up to three levels deep. This is used in large systems to organize the documentation into chapters that match the software's subsystems. For example, if your system is divided into subsystems, you will want to make a separate chapter for each subsystem. If the system is composed to several nested subsystems, then you will want to create nested chapters.

Header File - This layout item selects all of the classes, structs, enums, and functions, declared (usually the header files) in the specified file into the parent chapter. The symbols are assigned to a file when they are declared, not when they are implemented.

Class Name - This layout item includes only the named class into the parent chapter. This is useful if you have a set of private classes in a subsystem that should not be documented. The public classes can be explicitly specified, leaving the private classes undocumented. The header files should be specified in this field.

Function Name - This layout item includes only the named function into the parent chapter. This is useful if you have a set of private functions in a subsystem that should not be documented. For example, the public functions can be explicitly specified, leaving the private functions undocumented. If the function is overloaded, then all of the functions are documented. The return type and parameter list should not be specified.

Document Inserted - This layout item inserts the external document into the parent chapter. The document must be an ASCII text file. The text of the document is included directly into the document. The file is auto formatted the same way the source code comments are.

Document Linked - This layout item links, or references, the external document into the parent chapter. The document can be any format. A link is added to the HTML file and a reference is made in the document in the RTF file.

For more information about the layout window, press here.

The Comment Filter Window

This window lists the comment filter tags that are setup for the current configuration file. The comment tags are used to format and filter the source code comments into a form that is suitable for an external document. Tags can be create, edited, and deleted from this window.

For more information about the comment window, press here.

The Comment Filter Edit Window

Object Outline adapts to your project comment style. This window allows you to define the tags that the Object Outline filter uses. The tags are used by the comment filter to divide the source code comment into sections of text. The divided sections are then filtered differently depending upon the tags that you define in this window. Regular expressions can be used to reduce the number of tags defined. For more information see an example.

Tag Name - This is the name of the comment tag: "Example, Purpose, and Return" are common tag names. Regular expressions are allowed to be used to specified groups of tags.

Tag Filter Options:

> "Do Not Remove Tag From Comments" - This tells Object Outline, that yes this is a tag, but do not remove it from the comment. This is used to terminate an adjacent tag that removes comment text.
>
> "Always Remove Tag From Comments" - This tells Object Outline that the tag is not useful and should be unconditionally removed from the comment. Purpose tags often fall under this category.
>
> "Remove Tag Only When Empty" - This tells Object Outline, that the tag should only be removed if there is only white space between this and the next tag. This is used for removing tags that are in your comment template but are not used in most functions. Many commenting conventions require the used of a standard comment template. However, most functions do not require all of the sections in the template. This option allows the removal of those unused comment tags.

Tag Operations:

> "Remove Text After Tags" - This option, when selected, filters out the text between the current tag and the next tag. This is used to filter out function names and function prototypes that are redundant with the information that is generated by the Object Outline parser. It can also be used to not remove parts of comments that should not be seen by the documentation users.
>
> "Remove Text Before Tags" - This option filters out the text between the current tag and the last tag. This is used to filter out function names and function prototypes that are redundant with the information that is generated by the Object Outline parser.

For more information about regular expressions, press here.
For more information about the comment window, press here.

The Comment Filter Regular Expression Syntax

Standard wild cards can be used to specify comment tags. The regular expression rules are as follows:

1. Any character that is not a special character (to be defined) matches itself.

2. A backslash followed by any special character matches the literal character itself. The backslash escapes the special character.

3. The control characters are: + * ? . [ ] ^ $

4. The period matches any character except the new line.

5. A set of characters enclosed in brackets is a one character regular expression that matches any of the characters in that set. A range of characters can be indicated with a dash. However, if the first character of the set is a caret, then the regular expression matches any character except those in the set. The caret loses its special meaning if it is not the first character of the set.

6. A one character regular expression followed by an asterisk matches zero or more occurrences of the regular expression.

7. A one character regular expression followed by a plus matches one or more occurrences of the regular expression.

8. A question mark is an optional element. The proceeding regular expression can occur zero or once in the comment - no more.

Finally, the entire regular expression can be anchored to match only the beginning or end of a line.

1. If the caret is at the beginning the regular expression, then the matched string must be at the beginning of the line.

2. If the dollar sign is at the end of the regular expression, then the matched string must be at the end of the line.

The following escape codes can be used to matched control characters:

\b	backspace
\e	escape
\f	form feed
\n	new line
\r	carriage return
\t	tab

Defines needed for popular WIN32 C++ compilers.

Microsoft C/C++	`_WIN32,_MSC_VER=1000,_M_IX86=400,_WINDOWS,_MBCS,_MT`