Natural Docs 4 Matlab

Developer's guide

Natural Docs 4 Matlab is a documentation generator for Matlab language developed using the existing Natural Docs software (http://www.naturaldocs.org/) developed by Greg Valure.

This document tries to list all changes done to the original version of Natural Docs by explaining our process of development. That could be useful in order to develop another "custom" version of Natural Docs for another language.

Natural Docs 4 Matlab – How to try to make Natural Docs better?

This document tries to explain how we made Natural Docs 4 Matlab using the original Natural Docs developed by Greg Valure. We'll try to explain the basis of Natural Docs that are useful in order to adapt it to your own language, like we did for Matlab.

First of all, I'd like to thank and congratulate Greg Valure for its amazing job making Natural Docs. Having worked on it for a few weeks, WE can tell that he did a great work, and I'm going to try to save you some time if you want to « enhance » Natural Docs 1.4 (and 1.5).

The basis of Natural Docs – How it works

I'm going to try to keep it simple and just tell you the basis of what you need to know. Everything that is done before parsing won't be discussed here.

When you use Natural Docs to generate code, it parses your code. First of all, it divides the code from a given file into Comments (all comments lines) and Tokens (See NaturalDocs::Language::Advanced->ParseForCommentsAndTokens()).

Like Natural Docs documentation says, Tokens are defined as:

- All consecutive alphanumeric and underscore characters.

- All consecutive whitespace.

- A single line break. It will always be "\n"; you don't have to worry about platform differences.

- A single character not included above, which is usually a symbol. Multiple consecutive ones each get their own token.

Tokens and Comments are then parsed in their own way to form Topics. A Topic basically is whatever you want to document. A Topic has a type (function, variable, class…), title (the name), a body (the documentation of the topic), etc… (See NaturalDocs::Parser::ParsedTopics). I'll deal with this specific point later.

Once Topics have been generated for the whole file, they are sent to NaturalDocs::Builder::HTMLBase in order to generate the documentation. When it's done, parsing starts for the next file.

The module – What it does, what it doesn't

Whereas comments are auto-parsed by the existing code, Tokens have to be handled by the language module (inherited from NaturalDocs::Languages::Advanced). That might be the first thing you want to develop in order to have a better language support. Greg Valure did a very great job with the C# module, so it's very easy to use it as a model. To implement a module in the project, just add an entry in the Languages.txt file (See Customizing languages), and add a "use" statement for your module in NaturalDocs::Languages::Advanced.

What your module needs to do is very basic: parse Tokens (which basically are "extracts" of code), find and generate Topics. Most of Low Level Parsing Functions developed by Greg Valure in the C# module can be re-used for your module, with of course some changes to adapt to your language.

But the basic idea of the module is for you to develop functions that will find and generate a Topic for classes, functions, variables, etc… That means you should know every possible trick in the language.

When it's done, if your code is already documented for Natural Docs basic language support, you won't see many differences in the generated documentation. First, you'll have a nice and very useful list of super/sub-classes at the top of your page, and if you activate JavaDoc support (See attributes 4 and 5 of ND::Language::Advanced->ParseForCommentsAndTokens()), you won't have to specify the type of topic you're in, nor its name, while documenting your code.

At this point, we wanted some more things in our documentation, things that weren't doable just using our module:

- See inherited methods from every superclass
- See modifiers for variables, functions, etc…
- Sort variables, functions, etc… by name
- Show colored code in documentation
- …

Like WE said, that couldn't be done just using our module. So we took the decision of going deeper into Natural Docs, and see how we could do this, using as much as possible everything that was already developed, and modifying the less possible already existing code. The ultimate goal was for our addition to be compatible with every language supported by Natural Docs, so that we could release it as a "plugin", but that has not been done.

This documentation will now explain what changes we made in Natural Docs that lead to Natural Docs 4 Matlab.

Inherited Methods – The inheritance problem

The goal here was for a given class, to see in its documentation all methods inherited from its super-classes.

Like we said before, Natural Docs generate documentation “file by file”, i.e. it parses one file, generates Topics for it, generates HTML documentation, and starts over, deleting all the last file’s Topics (See NaturalDocs ::Builder->Run() line 215 : parse, generate, and start over !).

That mean you only store topics of the file you’re working on. This choice is understandable, because it reduces memory costs for big projects, but we choose to change things here.

To get through this, we did it as simple as possible: instead of doing « parse > generate > start again », we just did « parse > store parsed file into an array > start again until we have parsed everything > generate each entry of our array ».

To make that, the only change needed is right where the transition between parsing and documentation is done: NaturalDocs ::Builder->Run() line 215.

With that being done, we decided to add a new “step” before generating documentation. That’s where we’d be able to work with all the parsed files at the same time. To do that, we developed a new class: TopicTools. You have to think of it as a third part of the building process. In Natural Docs, it’s “Parse > Generate”, with our new class, it’s “Parse > Work on parsed files > Generate”.

That means we don’t interfere with existing code. Parsing still does parsing, and HTML generation still does HTML generation. Our work is in between.

On this class you can do many things. We first decided to develop the whole inheritance thing. Now that we have access to superclasses, everything’s easier. You just have to add Topics to a class, which are copy of superclasses’ Topics you want to have, or anything else you want, and HTML generation will generate it just as any Topic.

We decided to do our sorting in here, right before working on inheritance, so topics would already be sorted while transferred to subclasses.

We also add a function that lightens the summary if you have too many entries: for more than 20 entries in the same group, we add a new group topic for each first letter of topic’s title.

So once you managed to get all parsed files before the HTML generation, you can do pretty much everything you want with them. The good thing is also that you work on Topics, and those are independent of the language you work on. In our case, Matlab language is very “light” (meaning that the only TopicTypes we used are classes, attributes and functions), so what we develop won’t support every language, but the thing is that this class could be done to work with any language.

But the handling of inheritance, using all parsedFiles at once, prevents you from using the “updating documentation” feature of NaturalDocs, so you have to force rebuild, each time NaturalDocs is used. This is done in NaturalDocs::Settings->ParseCommandLine() by removing the handling of the –ro and –r options, and auto-calling NaturalDocs::Project->ReparseEverything() and NaturalDocs::Project->RebuildEverything().

Getting Modifiers & Code

A thing we also wanted to do was to be able to document modifiers automatically (We’ll talk about the code later).

The first thing you have to do is storing modifiers in your module. When you parse, you have to get them.

With that being done, you also have to store them with the Topic you create. At this point, we decided to add a new “attribute” to our Topics (NaturalDocs::Parser::ParsedTopic). The great thing about this is that programming in Perl allows you to have undefined attribute, so all existing Topics (especially those generated from comments) will still be ok.

Natural Docs, once it has parsed code and comments to create Topics, tries to merge them (NaturalDocs::Parser->MergeAutoTopics()). That’s an important part of the parsing process, but you also have to make some change in here, because if a topic has been generated using both comments and code, Natural Docs uses Topic created by comments and ignores (partly) the one created using code. So you’ll have to specify that you want to use modifiers stored in the Topic generated by code.

Obviously, this won’t display your modifiers in the documentation, you’ll also have to add in the HTML generation (NaturalDocs::Builder::HTMLBase) that you want to display modifiers. The two main functions you may want to change are BuildContent and BuildSummary. Obviously, you can also create a group topic for each modifier, and sort functions by modifiers, using your TopicsTools class.

For the code, it’s slightly different. The idea is the same, you have to add an attribute to ParsedTopic, make some changes in MergeAutoTopics(), etc… But getting code while parsing is more difficult if, like us, you want your code with your comments. Because as you know, the first stage of parsing in Natural Docs divides file in code and comments. So when you parse code, you obviously don’t have comments.

Here it gets a little ugly, but our goal was to use as much as possible Natural Docs existing functions, and modify at least as possible existing code. Let’s get back to our NaturalDocs::Language::Advanced->ParseForCommentsAndTokens() function. It uses to arrays: one for comments, one for tokens. Our idea was to add a third one, and store everything just like the one that stores tokens, except when we find a comment line.

When we're in that case, tokens' array stores a "\n", and comments' array stores the line. So our new array will store the comment line. With that, we'll be able to have comments, and keep the same indexing as the tokens' array, as comment line and "\n" are both stored in one index of the array. You also have to handle multi-comment, etc…

This new array now is the return value of the function, so that it doesn't have any effect on previous call of this function. When you want to get your code, you just have to get the return value, and that is done in your module. When you parse your tokens, you have the index of the current position, so you can easily have the starting and ending position of a code. And since tokens' and tokens+comments' array have the same indexing, you can now get your code using the start index and end index of your function.

That also is at this point that we decided to color our code. Coloration is done directly in HTML code since NDMarkup doesn't handle color. This is not very clean, and it would certainly be better to add color tags to NDMarkup and change the NaturalDocs::Builder::HTMLBase->NDMarkupToHTML() function so that it handles those tags.

Like Modifiers, you have to change things in NaturalDocs::Builder::HTMLBase so that it displays code. This can be done in BuildContent(). We added a function to display our code like we wanted, in spite of using NaturalDocs::Builder::HTMLBase->NDMarkupToHTML(). At this point you can also use some JavaScript or CSS to make the code show/hide, because it may overload your page.

One more thing for the code, we added an option in the line of command so that you can allow code to appear in documentation or not. This can be done by adding an option in NaturalDocs::Settings->ParseCommandLine().

So, in the end, what’s new in the code?

Well, first of all, we added new attributes to topics, to get easy access to a topic’s code, modifiers and superclasses. Using Perl, that doesn’t have any consequence for the other languages.

We also changed structure of the file treatment, so that we store all parsed files before generating documentation. That doesn’t have any effect either.

But what has is the new class: TopicsTools that is used just before generation. It’s really done just for Matlab, so if you want to do something like we did for Matlab, but for another language, that’s where you have to work. First: the module, then: those tools.

Like we said, TopicsTools uses Topics, so it should be possible to develop a version of it that would be compatible with every language, but our first goal was to develop a Matlab module for Natural Docs, so everything else we did was just some “extras”, and our goal was in no way to redevelop Natural Docs, we just wanted to try to change some things to make it more like we wanted it to be.

In the current state, Natural Docs 4 Matlab code is not optimized, pretty ugly at some places, but it works. That’s so because we tried to do as much as we could using what already exists in Natural Docs. We have a few ideas of « how to make it nicer and better », but that would take time.

The development of Natural Docs 4 Matlab has been done by Bertrand Richard for INRETS, from the previous work of Greg Valure.

Natural Docs official website: http://www.naturaldocs.org/

Contact Natural Docs 4 Matlab developers:

- bertrand.richard.insa@gmail.com
- Damien.sornette@inrets.fr

Changelog

Almost all changes from 1.5 version have been added to ND4M except the syntax highlighting.

Config\Languages.txt

- Added Matlab entry, with Full Languages Support.

Styles\

- Changed CSS code to add a horizontal scrollbar to the menu in HTML if necessary

NaturalDocs::Builder -> Run()

- Added %parsedFiles to store all parsedFile
- Added call to NaturalDocs::TopicsTools->WorkOnParsedFile()

NaturalDocs::Builder::FramedHTML->BuildFile()

- Added JavaScript code to hide / show code from documentation

NaturalDocs::Builder::HTML->BuildFile()

- Added JavaScript code to hide / show code from documentation
- Removed DocType declaration that created a conflict with code coloration

NaturalDocs::Builder::HTMLBase

- Added FormatCode() function to display Code the way we want

NaturalDocs::Builder::HTMLBase->BuildContent()

- Added Modifiers display in body
- Added Code display

NaturalDocs::Builder::HTMLBase->BuildClassHierarchy()

- Changed code to get rid of the 4 super/sub-classes display limit.

NaturalDocs::Error->HandleDeath()

- Changed text about getting help about the program after an error so that it refers to us instead of Greg Valure.

NaturalDocs::Languages

- Added a “use” statement for our module

NaturalDocs::Languages::Advanced->ParseForCommentsAndTokens()

- Added @tokensAndComments as a return value
- Changed code to handle @tokensAndComments the way we wanted
- Changed code to handle single line comment followed by multiline comment

NaturalDocs::Languages::Advanced->TokenizeLine()

- Changed code to handle @tokensAndComments

NaturalDocs::Languages::Advanced->TryToSkipString()

- Added a new constraint when token equals "\\", it caused problems when we had something like « '\' »

NaturalDocs::Languages::Base->ParsePrototype()

- Changed code to better handle declaration using spaces (i.e. deleted spaces before storing prototype)

NaturalDocs::Languages::Matlab

- Created the class and functions
- See NaturalDocs::Languages::CSharp for a well documented example, or "The Module – What it does, what it doesn't" part of this document

NaturalDocs::Parser::Native->ParseComment()

- Changed code to be able to parse comment even if the declaration of the comment ("keyword: title") doesn't have name. This was handled in NaturalDocs::Parser->MergeAutoTopics() but not before (in the parsing stage)

NaturalDocs::Parser::Native->ParseHeaderLine()

- Changed code to be able to parse comment even if the declaration of the comment ("keyword: title") doesn't have name. This was handled in NaturalDocs::Parser->MergeAutoTopics() but not before (in the parsing stage)

NaturalDocs::Parser->MergeAutoTopics()

- Changed code to handle new topics' attributes

NaturalDocs::Parser->OnComment()

- Changed code to be able to parse comment even if the declaration of the comment ("keyword: title") doesn't have name. This was handled in NaturalDocs::Parser->MergeAutoTopics() but not before (in the parsing stage)

NaturalDocs::Parser::ParsedTopic

- Added attributes to parsedTopics, plus getters and setters
- Added description about how modifiers and parents must be used/stored

NaturalDocs::Parser::ParsedTopic->New()

- Changed code to handle Modifiers and Parents storage.

NaturalDocs::Settings

- Added variable that handle code showing / hiding in documentation, plus getter and setter

NaturalDocs::Settings->ParseCommandLine()

- Added code to handle the new “-code” option in the command line
- Automatically call NaturalDocs::Project->ReparseEverything() and NaturalDocs::Project->RebuildEverything() (we don’t handle –r and –ro options anymore)

NaturalDocs::TopicsTools

- Created the class and functions