# Building a Website with Make4ht: Code, Minted, and Pygments

David Friant

March 29, 2026

**Abstract**

Syntax highlighting for code listings is a a critical feature of any website which needs to display large amounts of code. To that end, the Minted package for LaTeX provides excellent support for it in the generation of pdf documents. However, the default output of Make4ht when using it is in need of some modification. Minted is based upon the the Pygments Python library which will be called directly in the customize of the syntax highlighting process.

**Key Words:** LaTeX, Make4ht, Code, Syntax Highlighting, IndieWeb

## Intent and Purpose

With the mighty task of translating tables from LaTeX to HTML finished to an acceptable level for the moment, one turns their attention to code and, specifically, syntax highlighting. Code can certainly be displayed with ease by simply translating the LaTeX `verbatim` environment into an HTML `<pre>` block, however, this will deprive readers of the base quality of life provided by even simple syntax highlighting in both the PDF and HTML documents. Fortunately, this is a long-solved problem, allowing for a reasonably straightforward path towards implementing this feature.

## Minted

There is no shortage of tools available online to perform syntax highlighting for code blocks. However, given that these documents start as LaTeX files, the best option is to use Pygments[3]. Indeed, the `Minted`[2] package for LaTeX already utilizes it for code highlighting in PDF documents, thus solving half of the problem by simply adapting oneself to its use. There is a caveat, however, in that the implementation of code highlighting presented here uses only a subset of `Minted`'s commands for reasons to be discussed below.

Specifically, only the `\mintinline` and `\inputminted` commands will be allowed. The use of these is demonstrated in Listing 1. As can be inferred from the names and the example, the `\mintinline` command allows for the inclusion of a code snippet with syntax highlighting in a block of text without breaking while the `\inputminted` command creates a block of code which is included from a separate file.[A] Readers may note that, despite using the `\mintinline` command in these documents, there is no syntax highlighting for inline code on the HTML documents. This is purely a stylistic choice to prevent the text from becoming too busy. A small modification to the CSS file which styles this website (see this article), would highlight all the code snippets at once.

Listing 1: A simple demonstration of the two allowed commands from `Minted` package.

```
0 The entrance point to a C++ program is the \mintinline{cpp}|main()| function as
1 demonstrated in the code block below:
2
3 \inputminted[linenos,firstnumber=0]{cpp}{code/Example.cpp}
4
```



[A] One could even include the source code for their LaTeX document into the document itself.

As was the case with tables, one must unfortunately overwrite the default behavior of Make4ht in the use of the `Minted` package due to some undesirable output. This is best seen by examining the output from the processing of a simple block of code, the input of which is provided by Listing 2.

Listing 2: A simple example of C++ code to act as a point of discussion.

```
#include <iostream>

/**
 * The main entrance point of the program.
 */
int main(int argc, char* argv[]){
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

Listing 3 provides the output from the processing of Listing 2.[B] Undesirably, the default output of Make4ht includes spurious anchor tags at the start of every line and, more importantly, gives every token[1] a unique color. Normally, the CSS file generated in the output of Make4ht would define each color, potentially duplicating the definition of the color many times. This is unideal for obvious reasons. Thus, The remainder of this article will be dedicated to describing a manner in which to largely side-step Make4ht and instead call upon Pygments directly.

Listing 3: The default output of Make4ht leaves something to be desired as not only does it not assign each token a class as would be expected of a lexer, it gives every token a unique identifier instead.

```
<pre class='fancyvrb' id='fancyvrb1'><!--
     --><a id='x1-3r1'></a><!--
     --><span id='textcolor1'><!--
         --><span class='cmitt-10'>#</span><!--
     --></span><!--
     --><span id='textcolor2'><!--
         --><span class='cmitt-10'>include</span><!--
     --></span><!--
     --><span id='textcolor3'> </span><!--
     --><span id='textcolor4'><!--
         --><span class='cmitt-10'>&lt;iostream&gt;</span><!--
     --></span><!--
     --><a id='x1-5r2'></a>
        <a id='x1-7r3'></a>
<!-- --><span id='textcolor5'><!--
     --><span class='cmitt-10'>/**</span><!--
     --></span><!--
     --><a id='x1-9r4'></a>
<!-- --><span id='textcolor6'><!--
         --><span class='cmitt-10'> * The main entrance point of the <!--
         -->program.</span><!--
     --></span> <!--
     --><a id='x1-11r5'></a>
<!-- --><span id='textcolor7'><!--
         --><span class='cmitt-10'> */</span><!--
     --></span><!--
     --><a id='x1-13r6'></a>
<!-- --><span id='textcolor8'><!--
         --><span class='cmtt-10'>int</span><!--
```

[B] In order to respect the maximum allowed width for code listings, it has been necessary to break some lines across multiple and include a number of comments to prevent the injected line-feeds from affecting the format.

```
29        --></span><!--
30        --><span id='textcolor9'> </span><!--
31        --><span id='textcolor10'><!--
32            --><span class='cmtt-10'>main</span><!--
33        --></span><!--
34        --><span class='cmtt-10'>(</span><!--
35        --><span id='textcolor11'><!--
36            --><span class='cmtt-10'>int</span><!--
37        --></span><!--
38        --><span id='textcolor12'> </span><!--
39        --><span class='cmtt-10'>argc,</span><!--
40        --><span id='textcolor13'> </span><!--
41        --><span id='textcolor14'><!--
42            --><span class='cmtt-10'>char</span><!--
43        --></span><!--
44        --><span id='textcolor15'><!--
45            --><span class='cmtt-10'>*</span><!--
46        --></span><!--
47        --><span id='textcolor16'> </span><!--
48        --><span class='cmtt-10'>argv[]){</span><!--
49        --><a id='x1-15r7'></a>
50    <!-- --><span id='textcolor17'>    </span><!--
51        --><span class='cmtt-10'>std</span><!--
52        --><span id='textcolor18'><!--
53            --><span class='cmtt-10'>:</span><!--
54        --></span><!--
55        --><span id='textcolor19'><!--
56            --><span class='cmtt-10'>:</span><!--
57        --></span><!--
58        --><span class='cmtt-10'>cout</span><!--
59        --><span id='textcolor20'> </span><!--
60        --><span id='textcolor21'><!--
61            --><span class='cmtt-10'>&lt;</span><!--
62        --></span><!--
63        --><span id='textcolor22'><!--
64            --><span class='cmtt-10'>&lt;</span><!--
65        --></span><!--
66        --><span id='textcolor23'> </span><!--
67        --><span id='textcolor24'><!--
68            --><span class='cmtt-10'>"</span><!--
69        --></span><!--
70        --><span id='textcolor25'><!--
71            --><span class='cmtt-10'>Hello, World!</span><!--
72        --></span><!--
73        --><span id='textcolor26'><!--
74            --><span class='cmtt-10'>"</span><!--
75        --></span><!--
76        --><span id='textcolor27'> </span><!--
77        --><span id='textcolor28'><!--
78            --><span class='cmtt-10'>&lt;</span><!--
79        --></span><!--
80        --><span id='textcolor29'><!--
81            --><span class='cmtt-10'>&lt;</span><!--
82        --></span><!--
83        --><span id='textcolor30'> </span><!--
84        --><span class='cmtt-10'>std</span><!--
85        --><span id='textcolor31'><!--
86            --><span class='cmtt-10'>:</span><!--
87        --></span><!--
```

```
      --><span id='textcolor32'><!--
          --><span class='cmtt-10'>:</span><!--
      --></span><!--
      --><span class='cmtt-10'>endl;</span><!--
      --><a id='x1-17r8'></a>
<!-- --><span id='textcolor33'>    </span><!--
      --><span id='textcolor34'><!--
          --><span class='cmtt-10'>return</span><!--
      --></span><!--
      --><span id='textcolor35'> </span><!--
      --><span id='textcolor36'><!--
          --><span class='cmtt-10'>0</span><!--
      --></span><!--
      --><span class='cmtt-10'>;</span><!--
      --><a id='x1-19r9'></a>
<!-- --><span class='cmtt-10'>}</span>
</pre>
```

Similar to how the behavior of the `multirow` package was overwritten in the previous article, one should create a new file: `~/texmf/tex/latex/minted/minted.4ht`. The contents of this file are provided by Listing 4. Starting from the top, one first declares the requirement of the `shellesc` package which allows the Make4ht program to call outside programs. Next, a counter is set up to track the number of minted commands called. The `\processmintedinline` command is defined which calls a python script that handles the syntax highlighting and outputs the result to a file named using the counter. As is customary, a stub is created for the post-processing step. The definition of the `\mintinline` command itself is constrained to preparing the input parameters for processing via some regex substitutions. The definition of the `\inputminted` command is rather more straightforward as there is no need for the regex preprocessing.

Listing 4: The contents of the minted.4ht file for overwriting the behavior of the package.

```
\RequirePackage{shellesc}

\newcounter{mintedcounter}
\setcounter{mintedcounter}{0}

\NewDocumentCommand{\processmintinline}{O{} m m}{
    \ShellEscape{
            python3 ~/texmf/tex/latex/minted/mintinline.py \
            "#1" "#2" "#3" > _minted/\themintedcounter_PygmentsOutput.txt
    }
    \HCode{
            <stub class="inlinecode" file="\themintedcounter_PygmentsOutput.txt"
            lang="#2">\themintedcounter_PygmentsOutput.txt</stub>
    }
    \stepcounter{mintedcounter}
}

\ExplSyntaxOn
\RenewDocumentCommand{\mintinline}{O{} m v}{
    \tl_set:Nn \l_tmpa_tl {#3}
    \regex_replace_all:nnN {\\}{\\\\} \l_tmpa_tl
    \regex_replace_all:nnN {"}{\\"} \l_tmpa_tl
    \regex_replace_all:nnN {`}{\\`} \l_tmpa_tl

    \processmintinline[#1]{#2}{\l_tmpa_tl}
}
\ExplSyntaxOff
```

```
\Configure{mintinline}{}{}

\NewDocumentCommand{\inputminted}{O{} m m}{
    \ShellEscape{
            python3 ~/texmf/tex/latex/minted/inputminted.py \
            "#1" "#2" "#3" > _minted/\themintedcounter_PygmentsOutput.txt
    }
        \HCode{
                <stub class="codeblock" file="\themintedcounter_PygmentsOutput.txt"
                lang="#2">\themintedcounter_PygmentsOutput.txt</stub>
        }
    \stepcounter{mintedcounter}
}
```

## Pygments

The python scripts called in Listing 4 should be located in the same directory as the `minted.4ht` file. These simply get the appropriate lexer for a given language and format the output for HTML. The contents of the `inputminted.py` file are provided in Listing 5 to provide an example of the process.

Listing 5: The contents of the `inputminted.py` file.

```
import sys
from pygments import highlight
from pygments.lexers import (get_lexer_by_name)
from pygments.formatters import HtmlFormatter

class CodeHtmlFormatter(HtmlFormatter):
    def wrap(self, source):
        return self._wrap_code(source, include_div=False)
    #

    def _wrap_code(self, source, *, include_div):
        yield 0, '<pre class=\'' + sys.argv[2] + '\'><code>'
        for i, t in source:
            if i == 1:
                # it's a line of formatted code
                t += '</code><code>'
            #
            yield i, t
        #
        yield 0, '</code></pre>'
    #
#

def main():
    # argv[2] = lexer name
    # argv[3] = file path
    file = open(sys.argv[3], "rt")
    text = file.read()
    file.close()

    lexer = get_lexer_by_name(sys.argv[2])
    #from pygments.formatters import HtmlFormatter
    print(highlight(text, lexer, CodeHtmlFormatter()))
#

if __name__ == "__main__":
    main()
#
```

As can be observed, the code to call the Pygments module and even slightly customize it is quite simple. Starting from the `main()` function, the code reads in the target file to be highlighted, gets the appropriate lexer[5] for the language of that file, and calls the imported `highlight()` command. The formatter[4] provided to the `highlight()` command is a slightly modified instance of the basic `HTMLFormatter` provided by Pygments.

The customization of the formatter is very slight. Indeed, it simply wraps the contents of the entire file in a `<pre>` block and then each line of the file in `<code>` blocks. Listing 6 provides the output[C] of this customized process when applied to Listing 2. Compared to Listing 3, the customized output provides information on the language in the class attribute of the `<pre>` tag, removes the spurious `<a>` tags, and assigns each `<span>` tag a class associated with one of Pygments' builtin token types[6]. This allows CSS styling of the text per token class and per language, if it is thought necessary.

Listing 6: The code of Listing 2 processed by the customized system. Compare with Listing 3.

```
<pre class="cpp"><!--
     --><code><!--
         --><span class="cp">#include</span><!--
         --><span class="w"> </span><!--
         --><span class="cpf">&lt;iostream&gt;</span><!--
     --></code>
<!-- --><code><!--
     --></code>
<!-- --><code><!--
         --><span class="cm">/**</span><!--
     --></code>
<!-- --><code><!--
         --><span class="cm"> * The main entrance point of the program.</span><!--
     --></code>
<!-- --><code><!--
         --><span class="cm"> */</span><!--
     --></code>
<!-- --><code><!--
         --><span class="kt">int</span><!--
         --><span class="w"> </span><!--
         --><span class="nf">main</span><!--
         --><span class="p">(</span><!--
         --><span class="kt">int</span><!--
         --><span class="w"> </span><!--
         --><span class="n">argc</span><!--
         --><span class="p">,</span><!--
         --><span class="w"> </span><!--
         --><span class="kt">char</span><!--
         --><span class="o">*</span><!--
         --><span class="w"> </span><!--
         --><span class="n">argv</span><!--
         --><span class="p">[]){</span><!--
     --></code>
<!-- --><code><!--
         --><span class="w">    </span><!--
         --><span class="n">std</span><!--
         --><span class="o">::</span><!--
         --><span class="n">cout</span><!--
         --><span class="w"> </span><!--
         --><span class="o">&lt;&lt;</span><!--
         --><span class="w"> </span><!--
         --><span class="s">"Hello, World!"</span><!--
         --><span class="w"> </span><!--
```

[C] Again, the output has been broken over multiple lines, and comments have been added to maintain the format.

```
43          --><span class="o">&amp;lt;&amp;lt;</span><!--
44          --><span class="w"> </span><!--
45          --><span class="n">std</span><!--
46          --><span class="o">::</span><!--
47          --><span class="n">endl</span><!--
48          --><span class="p">;</span><!--
49      --></code>
50  <!-- --><code><!--
51          --><span class="w">    </span><!--
52          --><span class="k">return</span><!--
53          --><span class="w"> </span><!--
54          --><span class="mi">0</span><!--
55          --><span class="p">;</span><!--
56      --></code>
57  <!-- --><code><!--
58          --><span class="p">}</span><!--
59      --></code>
60  <!-- --><code></code>
61  </pre>
```

## Final Notes

The post-processing is simple enough that it will not be greatly expounded upon here. One must simply replace the stubs generated by the commands in Listing 4 with the text of the matching files. There should be no need for examples this time around; this series of articles has already clearly shown the system at work.

With this article finished, there remains only one major type of content to be implemented before this system is considered "complete": images of both the raster and vector varieties. The next article in this series describes just that.

## References


[1] Lexical tokens. `https://en.wikipedia.org/wiki/Lexical_analysis#Token`. Accessed: 2026-03-27.

[2] Minted. `https://ctan.org/pkg/minted`. Accessed: 2026-03-27.

[3] Pygments. `https://pygments.org`. Accessed: 2026-03-27.

[4] Pygments formatters. `https://pygments.org/docs/formatters`. Accessed: 2026-03-28.

[5] Pygments lexers. `https://pygments.org/docs/lexers`. Accessed: 2026-03-28.

[6] Pygments tokens. `https://pygments.org/docs/tokens`. Accessed: 2026-03-28.