

# Vesti Transpiler User Manual

Sungbae Jeong

November 25, 2025

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Structure of Vesti File</b>	<b>2</b>
<b>3</b>	<b>Keywords</b>	<b>2</b>
3.1	<code>docclass</code> keyword . . . . .	3
3.2	<code>importpkg</code> keywords . . . . .	3
3.3	<code>startdoc</code> keyword . . . . .	3
3.4	<code>useenv</code> keyword . . . . .	3
3.5	<code>begenv</code> and <code>endenv</code> keywords . . . . .	4
3.6	<code>defun</code> keyword . . . . .	4
<b>4</b>	<b>Builtins</b>	<b>4</b>
4.1	<code>#label</code> builtin . . . . .	5
4.2	<code>#eq</code> builtin . . . . .	5
4.3	<code>#chardef</code> builtin . . . . .	6
4.4	<code>#mathchardef</code> builtin . . . . .	6
4.5	<code>#get_filepath</code> builtin . . . . .	6
4.6	<code>#at_on</code> and <code>#at_off</code> builtins . . . . .	6
4.7	<code>#ltx3_on</code> and <code>#ltx3_off</code> builtins . . . . .	6
4.8	<code>#textmode</code> and <code>#mathmode</code> builtins . . . . .	7
<b>5</b>	<b>Lua API</b>	<b>7</b>
<b>6</b>	<b>Source Code of This Document</b>	<b>8</b>

## 1 Introduction

I used to create several documents using  $\text{\LaTeX}$  (or plain  $\text{\TeX}$  but  $\text{\TeX}$  is quite cumbersome to write—especially when working with very complex tables or inserting images). Its markdown-like syntax is also not comfortable to use. For example, here is a simple  $\text{\LaTeX}$  document:

```
1 % coprime is my custom class. See https://github.com/e0328eric/coprime.
2 \documentclass[tikz, geometry]{coprime}
3
4 \settitle{My First Document}{Sungbae Jeong}{}
5 \setgeometry{a4paper, margin = 2.5cm}
6
7 \begin{document}
8 \section{Foo}
9 Hello, World!
10 \begin{figure}[ht]
11   \centering
12   \begin{tikzpicture}
13     \draw (0,0) -- (1,1);
14   \end{tikzpicture}

```

```

15 \end{figure}
16
17 The code above is a figure using TikZ.
18
19 \end{document}

```

What annoys me most when using it is the ‘`\begin`’ and ‘`\end`’ blocks. Is there a way to write something much simpler? This question led me to start this project. Currently, the following code is generated into the `LATEX` code above (except comments) using vesti:

```

1 % coprime is my custom class. See https://github.com/e0328eric/coprime.
2 docclass coprime (tikz, geometry)
3
4 \settitle{My First Document}{Sungbae Jeong}{}
5 \setgeometry{a4paper, margin = 2.5cm}
6
7 startdoc
8
9 \section{Foo}
10 Hello, World!
11 useenv figure [ht] {
12     \centering
13     useenv tikzpicture {
14         \draw (0,0) -- (1,1);
15     }
16 }
17
18 The code above is a figure using TikZ.

```

Everywhere in this paper, the symbol  $\sqcup$  denotes a “space” when one use Vesti.

## 2 Structure of Vesti File

Vesti is similar as `LATEX`. Its structure consists with two parts: `preamble` and `main`. Preamble is the place where `LATEX` documentclass, packages, and several settings are located. Main body is where actual documentation is located. Below figure is the simple Vesti documentation.

```

1 docclass article (10pt)
2 importpkg {
3     geometry (a4paper, margin=2.2cm)
4 }
5 startdoc
6 Hello, Vesti!

```

We will see later, but the very difference with `LATEX` is that Vesti has its own keywords (keywords are colored with purple). It makes the code readable and it is easier and faster to write the document. The keyword `startdoc` splits the preamble and the main part of the documentation similar with

`\begin{document}` in `LATEX`. However, Vesti does not have the analogous part of `\end{document}`, because almost every `LATEX` document (99.999% I’m sure) does not have any code below `\end{document}`. For this reason, Vesti automatically ends document when EOF (End Of File) is found.

## 3 Keywords

Followings are reserved as keywords. In this document, every Vesti keyword has the form like `this`.

```

begenv      compty      cpfile      defenv
defun       docclass     endenv      importmod
importpkg   importves   startdoc   useenv

```

Table 1: Keywords in Vesti

### 3.1 `docclass` keyword

Keyword `docclass` is an analogous of `\documentclass` in L<sup>A</sup>T<sub>E</sub>X. If `docclass` is in the main paragraph, it acts just a normal word. In other words, `docclass` actives only in the preamble. The syntax of `docclass` is following:

```
docclass  $\sqcup$  <class name>  $\sqcup$  (<arguments>)
```

Here, arguments are separated by commas and embraced by `()`. Here are some examples.

- `docclass  $\sqcup$  article`
- `docclass  $\sqcup$  article  $\sqcup$  (10pt)`
- `docclass  $\sqcup$  article  $\sqcup$  (10pt,  $\sqcup$  twocols)`
- `docclass  $\sqcup$  article  $\sqcup$  (10pt,twocols)`

### 3.2 `importpkg` keywords

Keyword `importpkg` is an analogous of `\usepackage` in L<sup>A</sup>T<sub>E</sub>X. If `importpkg` is in the main paragraph, it acts just a normal word. In other words, `importpkg` actives only in the preamble.

`importpkg` has two different syntax. First one is same as `docclass`.

```
importpkg  $\sqcup$  <pkg-name>  $\sqcup$  (arguments)
```

Here, arguments are separated by commas and embraced by `()`. In the practical case, one should include several packages with options. `importpkg` also supports such case. We will look at an example instead of giving rigorous grammar.

```

1 importpkg {
2     amsmath, amssymb, amsthm,
3     geometry (a4paper, margin=2.2cm),
4 }
```

As one can see, inside of `{}`, several packages can be used together with their options.

### 3.3 `startdoc` keyword

Keyword `startdoc` tells to Vesti that the main document starts. In the main document, you can also write `startdoc` in the main document. In that case, `startdoc` does nothing.

### 3.4 `useenv` keyword

As the name implies, keyword `useenv` is an analogous of `\begin{...}` and `\end{...}` pair in L<sup>A</sup>T<sub>E</sub>X. The simplest `useenv` is like this.

```
useenv center {
    Hello, World!    or    useenv center { Hello, World! }
}
```

As you can see, `useenv center` is the part of `\begin{center}`, and the single `}` is the part of `\end{center}`. Since Vesti knows their pair, one can write a code with several environment, and each pair is properly matched. For instance, above example is written in Vesti like follows. Here, `\useenv` just prints `useenv` in that style.

```

1 useenv figure [ht] {
2   \centering
3   useenv tikzpicture {
4     useenv scope {
5       \path (0,0) node {\vbox{
6         %#\hbox{\tt\useenv center \{}}
7         %#\hbox{\tt\obeyspaces Hello, World!}
8         %#\hbox{\tt\obeyspaces\}}
9       }};
10    }
11    \path (2.3,0) node {or};
12    useenv scope [shift={(6,0)}] {
13      \path (0,0) node {\tt\useenv center \{ Hello, World! \}};
14    }
15  }
16 }
```

Full syntax about **useenv** is the following.

**useenv**  $\sqcup$  <environment name>  $\sqcup$  <argument><sup>\*</sup>  $\sqcup$  { <body> }

where '\*' means that the number of <argument> is zero or at least one, and

$$\text{<argument>} = \begin{cases} (\text{<argument>}) & \text{mandatory arguments} \\ [\text{<argument>}] & \text{optional arguments} \end{cases}$$

For instance, below one is a valid Vesti code (environment foo is undefined in general). As one can see, spaces cannot exist in between <argument>s.

```

1 useenv foo (asd)(fff)[\ames and \awdsa](askws)[\rrsaa] {
2   foobar
3 }
```

### 3.5 **begenv** and **endenv** keywords

As the name implies, both keywords **begenv** and **endenv** are analogous of `\begin{...}` and `\end{...}` pair in L<sup>A</sup>T<sub>E</sub>X, respectively. Thus below code

```

1 begenv center
2   asdsad
3 endenv
```

is exactly same as

```

1 useenv center {
2   asdsad
3 }
```

Then why we need **begenv** and **endenv** if we already have **useenv**? The only reason which both **begenv** and **endenv** exist is defining new environment. For instance, one defines a new environment like following.

```

1 defenv foo {\begenv minipage (\textwidth)\}{\endenv}
```

### 3.6 **defun** keyword

## 4 Builtins

Vesti also has its own builtin functions, which are prefixed with #. One might wonder what distinguishes builtins from keywords. In fact, from the compiler's internal perspective, there is no real difference. However,

in actual language usage, constantly typing the prefix can be somewhat tedious, especially for functions that are commonly used.

From the perspective of language design –particularly in Vesti– it is sometimes desirable to use names that cannot serve as keywords. For example, Vesti provides a built-in function `#label`, which will be explained later. Since Vesti is a typewriting-oriented language, the word “label” is often used in its ordinary sense rather than in its special semantic meaning within the language.

Followings are reserved as builtin functions.

```
#at_off      #at_on       #chardef     #enum        #eq
#get_filepath #label       #ltx3_off    #ltx3_on    #mathchardef
#mathmode     #noltx3    #picture     #raw_tex    #showfont
#textmode
```

Table 2: Builtins in Vesti

Since `fancyvrb` does not allow verbatim-like commands inside of `Verbatim` environment, builtins are not colored in example codes.

## 4.1 `#label` builtin

For the compiling issue, if one label some environment, one should write like follows using `LATEX` function `\label`.

```
1 useenv theorem { \label{eq:1}
2   1 + 1 = 2.
3 }
```

However, such code is not natural. Thus `#label` comes into the place. One can label environments using `#label` builtin like this.

```
1 #label(eq:1) useenv theorem {
2   1 + 1 = 2.
3 }
```

One can also write like the following, which I personally prefer.

```
1 #label(eq:1)
2 useenv theorem {
3   1 + 1 = 2.
4 }
```

## 4.2 `#eq` builtin

`#eq` is a abbreviation of ‘`useenv` equation’. Actually, below codes are same.

```
1 #label(eq:1)
2 useenv equation {
3   \sum_{n=1}^{\infty} {1/n^2} = {\pi^2/6}.
4 }
```

```
1 #eq(eq:1) {
2   \sum_{n=1}^{\infty} {1/n^2} = {\pi^2/6}.
3 }
```

Since Vesti is used in the mathematical documents mainly, small syntactic suger is needed, so Vesti introduce `#eq` builtin.

## 4.3 #chardef builtin

#chardef defines a text token using the unicode codepoint. Here is the grammar of #chardef builtin.

```
#chardef \<unicode-codepoint> \<function>
```

Here, <unicode-codepoint> must be hexadecimal.

## 4.4 #mathchardef builtin

#mathchardef defines a math token using the unicode codepoint. Here is the grammar of #mathchardef builtin.

```
#mathchardef \<kind> \<font-num> \<unicode-codepoint> \<function>
```

Here, <unicode-codepoint> must be hexadecimal. Before explaining each parameter, introduce some examples.

- #mathchardef \.opening \0 \29d8 \lfoo
- #mathchardef \.ordinary \0 \2202 \diff

## 4.5 #get\_filepath builtin

Because of the Vesti internal, if one includes picture by using \includegraphics for instance, providing the raw path cause an error from L<sup>A</sup>T<sub>E</sub>X with “file not found”. #get\_filepath builtin adjust raw file path into the new relative one which L<sup>A</sup>T<sub>E</sub>X knows. Here is the example code.

```
1 \includegraphics{#get_filepath(./foo.png)}
```

Internally, #get\_filepath changes file path into the relative one with respect to .vesti-dummy. For example, if foo.png, foo.ves and .vesti-dummy are located in the root directory, then inside of foo.ves, #get\_filepath(./foo.png) will changed into ../foo.png because for .vesti-dummy, foo.png is located outside of it.

## 4.6 #at\_on and #at\_off builtins

Both builtins are same as L<sup>A</sup>T<sub>E</sub>X functions \makeatletter and \makeatother plus changes Vesti lexer such that @ character is also can be a L<sup>A</sup>T<sub>E</sub>X function name. For instance, without using #at\_on command, below code does not compiles.

```
1 defun @foo (#1) {Hello, #1!}
```

This is because defun expects a valid L<sup>A</sup>T<sub>E</sub>X function name but the character @ is not a valid one except after the \makeatletter function. Since defun is a Vesti grammar, there is no way to tell Vesti that @ is a right one although one uses \makeatletter. By using #at\_on, below code then compiles.

```
1 #at_on
2 defun @foo (#1) {Hello, #1!}
3 #at_off
```

One can not using #at\_off, but I strongly recommand to match pairs.

## 4.7 #ltx3\_on and #ltx3\_off builtins

Those builtins are similar with #at\_on and #at\_off pairs but for L<sup>A</sup>T<sub>E</sub>X3.

## 4.8 #textmode and #mathmode builtins

Internally, vesti tracks so called `text mode` and `math mode`. In the `math mode`, for instance, the token `->` is changed into `\rightarrow` inside of the `math mode`. The idea is that Vesti changes `->` into `\to`, and so on. But when defining some function using `defun`, for example, one might want to define

```
1 useenv foo {-><-}
```

so that `$\foo$` makes  $\rightarrow\leftarrow$ , one should change the mode. Here, `#mathmode` comes into the place. One should write instead that

```
1 useenv foo {#mathmode{-><-}}
```

## 5 Lua API

Vesti uses Lua in both building script and inline inside of Vesti code. Below are functions which are baked in Vesti compiler.

compile	download	engineType	getCurrentDir
getModule	joinpath	mkdir	parse
print	setCurrentDir	unzip	vestiDummyDir

Table 3: Keywords in Vesti

## 6 Source Code of This Document

Below code was generated by inline lua.

```
1 docclass article (10pt)
2 importpkg {
3     geometry (a4paper, margin = 2.2cm),
4     xcolor,
5     tikz,
6     fancyvrb,
7 }
8
9 \title{Vesti Transpiler User Manual}
10 \author{Sungbae Jeong}
11
12 importves (font.ves)
13
14 % read file contents using lua
15 #lu:
16 local function read_all(path)
17     local f, err = io.open(path, "rb")
18     assert(f, ("cannot open %s: %s"):format(path, err))
19     local data = f:read("*a")
20     f:close()
21     return data
22 end
23 :lu#<readAll>
24
25 % definition of \keyword command
26 defun [!] keyword (m) {{\tt\color{purple}#1}}
27 defun [!] builtin (v) {{\tt\color{blue!65!yellow}\##1}}
28 #lu:
29 local commands = { "useenv", "begenv", "endenv", "defun", "defenv" }
30 for _, command in ipairs(commands) do
31     local s = string.format(
32         "defun %s (s) {\\"IfBooleanTF{#1}{\\keyword{%%-%s-%%}}{\\keyword{%%-%s-%%} }}",
33         command, command, command
34     )
35     vesti.print(vesti.parse(s))
36 end
37 :lu#
38
39 % space character used in the textbook.
40 #at_on
41 #chardef 2423 \@b
42 defun ob {\kern2pt{\tt\@b}\kern2pt}
43 #at_off
44
45 startdoc
46 \maketitle
47 \tableofcontents
48
49 \section{Introduction}
50 I used to create several documents using \LaTeX (or plain \TeX but \TeX is
51 quite cumbersome to write—especially when working with very complex tables or
52 inserting images). Its markdown-like syntax is also not comfortable to use. For
53 example, here is a simple \LaTeX document:
54
55 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%
56     % coprime is my custom class. See https://github.com/e0328eric/coprime.
```

```

57 \documentclass[tikz, geometry]{coprime}
58
59 \settitle{My First Document}{Sungbae Jeong}{}
60 \setgeometry{a4paper, margin = 2.5cm}
61
62 \begin{document}
63 \section{Foo}
64 Hello, World!
65 \begin{figure}[ht]
66   \centering
67   \begin{tikzpicture}
68     \draw (0,0) -- (1,1);
69   \end{tikzpicture}
70 \end{figure}
71
72 The code above is a figure using TikZ.
73
74 \end{document}
75 -%
76
77 What annoys me most when using it is the \lq\verb@\begin@\rq\ and
78 \lq\verb@\end@\rq\ blocks. Is there a way to write something much simpler? This
79 question led me to start this project. Currently, the following code is
80 generated into the \LaTeX code above (except comments) using vesti:
81
82 \useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-
83 % coprime is my custom class. See https://github.com/e0328eric/coprime.
84 +keyword|docclass@ coprime (tikz, geometry)
85
86 \settitle{My First Document}{Sungbae Jeong}{}
87 \setgeometry{a4paper, margin = 2.5cm}
88
89 +keyword|startdoc@
90
91 \section{Foo}
92 Hello, World!
93 +keyword|useenv@ figure [ht] {
94   \centering
95   +keyword|useenv@ tikzpicture {
96     \draw (0,0) -- (1,1);
97   }
98 }
99
100 The code above is a figure using TikZ.
101 -%
102
103 Everywhere in this paper, the symbol \ob\ denotes a \lq\lq space\rq\rq\ when
104 one use Vesti.
105
106 \section{Structure of Vesti File}
107 Vesti is similar as \LaTeX. Its structure consists with two parts: {\tt preamble} and
108 {\tt main}. Preamble is the place where \LaTeX\ documentclass, packages, and
109 several settings are located. Main body is where actual documentation is located.
110 Below figure is the simple Vesti documentation.
111
112 \useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-
113 +keyword|docclass@ article (10pt)
114 +keyword|importpkg@ {
115   geometry (a4paper, margin=2.2cm)
116 }

```

```

117 +keyword|startdoc@
118 Hello, Vesti!
119 -%}
120
121 We will see later, but the very difference with \LaTeX\ is that Vesti has its
122 own keywords (keywords are colored with purple). It makes the code readable and
123 it is easier and faster to write the document. The keyword startdoc splits
124 the preamble and the main part of the documentation similar with
125
126 % Don't ask why I chose Q for catcode 0.
127 %#{\tt\catcode`Q=0 Q\catcode`\\=12 \begin{document}Q}} in \LaTeX.
128 However, Vesti does not have the analogous part of
129 %#{\tt\catcode`Q=0 Q\catcode`\\=12 \end{document}Q},
130 because almost every \LaTeX\ document (99.999\% I'm sure) does not have any code
131 below %#{\tt\catcode`Q=0 Q\catcode`\\=12 \end{document}Q}.
132 For this reason, Vesti automatically ends document when EOF (End Of File) is
133 found.
134
135 \section{Keywords}
136 Followings are reserved as keywords. In this document, every Vesti keyword has
137 the form like \keyword{this}.
138 useenv table [ht] {
139     \centering
140     #lu:
141     local content = read_all("../src/lexer/Token.zig")
142
143     -- Lua's built-in patterns don't support lookahead.
144     -- We capture both the keyword and the TokenType, then filter out 'deprecated'.
145     -- Pattern breakdown:
146     --  %.%{          => matches ".{"
147     --  %s*"([""]+)" => a quoted string -> capture 1
148     --  %s*,%s*TokenType%.([%w_]++) => TokenType.<Name> -> capture 2
149     local pat = "%%.%{%.%{[%""]+)%"%"s*,%s*TokenType%.([%""]++)"
150
151     local keywords = {}
152     for name, tok in content:gmatch(pat) do
153         if tok ~= "deprecated" then
154             keywords[#keywords + 1] = name
155         end
156     end
157
158     table.sort(keywords)
159
160     vesti.print([[\begin{tabular}{cccc}]])
161
162     for i, kw in ipairs(keywords) do
163         local cell = string.format("\\\ keyword{%-s}", kw)
164         if (i % 4) == 0 then
165             vesti.print(cell .. [[\\]])
166         else
167             vesti.print(cell .. "&")
168         end
169     end
170
171     vesti.print([[\end{tabular}]])
172     :lu#[readAll]
173     \caption{Keywords in Vesti}
174 }
175
176 \subsection{\keyword{docclass} keyword}

```

```

177 Keyword \keyword{docclass} is an analogous of \verb|\documentclass| in \LaTeX.
178 If \keyword{docclass} is in the main paragraph, it acts just a normal word.
179 In other words, \keyword{docclass} actives only in the preamble.
180 The syntax of \keyword{docclass} is following:
181
182 \useenv center {
183   \keyword{docclass}\ob<class name>\ob{\tt()<arguments>\tt}
184 }
185 Here, arguments are separated by commas and embraced by {\tt ()}. Here are some
186 examples.
187
188 \goodbreak
189 \useenv itemize {
190   \item \keyword{docclass}\ob{\tt article}
191   \item \keyword{docclass}\ob{\tt article}\ob(10pt)
192   \item \keyword{docclass}\ob{\tt article}\ob(10pt,\ob twocols)
193   \item \keyword{docclass}\ob{\tt article}\ob(10pt,twocols)
194 }
195
196 \subsection{\keyword{importpkg} keyword}
197 Keyword \keyword{importpkg} is an analogous of \verb|\usepackage| in \LaTeX.
198 If \keyword{importpkg} is in the main paragraph, it acts just a normal word.
199 In other words, \keyword{importpkg} actives only in the preamble.
200
201 importpkg has two different syntax. First one is same as docclass.
202 \useenv center {
203   \keyword{importpkg}\ob<pkg-name>\ob{\tt()<arguments>\tt}
204 }
205 Here, arguments are separated by commas and embraced by {\tt ()}.
206 In the practical case, one should include several packages with options.
207 importpkg also supports such case. We will look at an example instead of
208 giving rigorous grammar.
209 \useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%
210 +\keyword{importpkg}@ {
211   amsmath, amssymb, amsthm,
212   geometry (a4paper, margin=2.2cm),
213 }
214 -%}
215
216 \noindent As one can see, inside of \verb|{}|, several packages can be used
217 together with thier options.
218
219 \subsection{\keyword{startdoc} keyword}
220 Keyword \keyword{startdoc} tells to Vesti that the main document starts. In the
221 main document, you can also write \keyword{startdoc} in the main document. In
222 that case, \keyword{startdoc} does nothing.
223
224 \subsection{\useenv keyword}
225 As the name implies, keyword \useenv is an analogous of \verb|\begin{...}| and
226 \verb|\end{...}| pair in \LaTeX.
227 The simplest \useenv is like this.
228
229 \useenv figure [ht] {
230   \centering
231   \useenv tikzpicture {
232     \useenv scope {
233       \path (0,0) node {\vbox{
234         %#\hbox{\tt\useenv center \{}%
235         %#\hbox{\tt\obeyspaces Hello, World!}
236         %#\hbox{\tt\obeyspaces\}}}

```

```

237     }};
238 }
239 \path (2.3,0) node {or};
240 useenv scope [shift={(6,0)}]{
241     \path (0,0) node {\tt\useenv center \{ Hello, World! \}};
242 }
243 }
244 }
245
246 As you can see, {\tt\useenv center} is the part of \verb|\begin{center}|, and
247 the single {\tt\}} is the part of \verb|\end{center}|. Since Vesti knows their
248 pair, one can write a code with several environment, and each pair is properly
249 matched. For instance, above example is written in Vesti like follows. Here,
250 \verb|\useenv| just prints \useenv in that style.
251
252 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%
253 |+color|purple@useenv@ figure [ht] {
254     \centering
255     |+color|purple@useenv@ tikzpicture {
256         |+color|purple@useenv@ scope {
257             \path (0,0) node {\vbox{
258                 |+color|blue@%#@\hbox{\tt\useenv center \{}}
259                 |+color|blue@%#@\hbox{\tt\obeyspaces Hello, World!}
260                 |+color|blue@%#@\hbox{\tt\obeyspaces\}}
261             }};
262         }
263         \path (2.3,0) node {or};
264         |+color|purple@useenv@ scope [shift={(6,0)}] {
265             \path (0,0) node {\tt\useenv center \{ Hello, World! \}};
266         }
267     }
268 }
269 -%}
270
271 Full syntax about \useenv is the following.
272 useenv center {
273     \useenv\ob<environment name>\ob<argument>*\ob{\tt\{} <body> {\tt\}}
274 }
275 where '*' means that the number of <argument> is zero or at least one, and
276 $$ 
277     "<argument>" = useenv cases {
278         "<argument>" & "mandatory arguments" \\
279         "[<argument>]" & "optional arguments"
280     }
281 $$ 
282 For instance, below one is a valid Vesti code (environment {\tt foo} is
283 undefined in general). As one can see, spaces cannot exist in between <argument>s.
284 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@] {
285 %#|+color|purple@useenv@ foo (asd)(fff)[\ames and \awdsa](askws)[\rrsaa] {
286 %#    foobar
287 %#}
288 }
289
290 \subsection{\begenv and \endenv keywords}
291 As the name implies, both keywords \begenv and \endenv are analogous of
292 \verb|\begin{...}| and \verb|\end{...}| pair in \LaTeX, respectively.
293 Thus below code
294 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@] {
295 %#|+color|purple@begenv@ center
296 %#    asdsad

```

```

297 | %#|+color|purple@endenv@
298 |
299 | is exactly same as
300 | useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@] {
301 | %#+keyword|useenv@ center {
302 | %#    asdsad
303 | %#}
304 |
305 | Then why we need \begenv and \endenv if we already have \useenv*?
306 | The only reason which both \begenv and \endenv exist is defining new
307 | environment.
308 | For instance, one defines a new environment like following.
309 | useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@] {
310 | %#+keyword|defenv@ foo {+keyword|begenv@ minipage (\textwidth){+keyword|endenv@}
311 |
312 |
313 | \subsection{\defun keyword}
314 |
315 | \section{Builtins}
316 | Vesti also has its own builtin functions, which are prefixed with \#.
317 | One might wonder what distinguishes builtins from keywords. In fact, from the
318 | compiler's internal perspective, there is no real difference. However, in actual
319 | language usage, constantly typing the prefix can be somewhat tedious, especially
320 | for functions that are commonly used.
321 |
322 | From the perspective of language design --particularly in Vesti-- it is
323 | sometimes desirable to use names that cannot serve as keywords. For example,
324 | Vesti provides a built-in function \builtin{label}, which will be explained
325 | later. Since Vesti is a typewriting-oriented language, the word \lq\lq
326 | label\rq\rq\ is often used in its ordinary sense rather than in its special
327 | semantic meaning within the language.
328 |
329 | Followings are reserved as builtin functions.
330 |
331 | useenv table [ht] {
332 |     \centering
333 |     #lu:
334 |     local content = read_all("../src/lexer/Token.zig")
335 |
336 |     -- match .{ "here" }
337 |     local pat = "%.%[%s*\\"([^\"]+)\\"%s*%]"
338 |
339 |     local builtins = {}
340 |     for name, tok in content:gmatch(pat) do
341 |         builtins[#builtins + 1] = name
342 |     end
343 |     table.sort(builtins)
344 |
345 |     vesti.print([[\begin{tabular}{ccccc}]])
346 |
347 |     for i, kw in ipairs(builtins) do
348 |         local cell = string.format("\\"builtin@%s@", kw)
349 |         if (i % 5) == 0 then
350 |             vesti.print(cell .. [[\\]])
351 |         else
352 |             vesti.print(cell .. "&")
353 |         end
354 |     end
355 |
356 |     vesti.print([[\\end{tabular}]])

```

```

357 :lu#[readAll]
358 \caption{Builtins in Vesti}
359 }
360 Since {\tt fancyvrb} does not allow verbatim-like commands inside of
361 {\tt Verbatim} environment, builtins are not colored in example codes.
362
363 \subsection{\tt builtin|label| builtin}
364 For the compiling issue, if one label some environment, one should write like
365 follows using \LaTeX\ function \verb|\label|.
366 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=&|@] {
367 %#&keyword|useenv@ theorem { \label{eq:1}
368 %#     1 + 1 = 2.
369 %#}
370 }
371
372 However, such code is not natural. Thus \tt builtin|label| comes into the place.
373 One can label environments using \tt builtin|label| builtin like this.
374 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=&|@] {
375 %%#label(eq:1) &keyword|useenv@ theorem {
376 %#     1 + 1 = 2.
377 %#}
378 }
379 One can also write like the following, which I personally prefer.
380 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=&|@] {
381 %%#label(eq:1)
382 %#&keyword|useenv@ theorem {
383 %#     1 + 1 = 2.
384 %#}
385 }
386
387 \subsection{\tt builtin|eq| builtin}
388 \tt builtin|eq| is a abbreviation of \lq\keyword{%-useenv-}%\ equation\rq.
389 Actually, below codes are same.
390
391 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@] {
392 %%#label(eq:1)
393 %#+keyword|useenv@ equation {
394 %#     \sum_{n=1}^{\infty} {1/n^2} = {\pi^2/6}.
395 %#}
396 }
397 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@] {
398 %%#eq(eq:1) {
399 %#     \sum_{n=1}^{\infty} {1/n^2} = {\pi^2/6}.
400 %#}
401 }
402 Since Vesti is used in the mathematical documents mainly, small syntactic suger
403 is needed, so Vesti introduce \tt builtin|eq| builtin.
404
405 \subsection{\tt builtin|chardef| builtin}
406 \tt builtin|chardef| defines a text token using the unicode codepoint.
407 Here is the grammar of \tt builtin|chardef| builtin.
408 useenv center {
409     \tt builtin|chardef| \ob<unicode-codepoint>\ob<function>
410 }
411 Here, <unicode-codepoint> must be hexadecimal.
412
413 \subsection{\tt builtin|mathchardef| builtin}
414 \tt builtin|mathchardef| defines a math token using the unicode codepoint.
415 Here is the grammar of \tt builtin|mathchardef| builtin.
416 useenv center {

```

```

417 \builtin|mathchardef|\ob<kind>\ob<font-num>\ob<unicode-codepoint>\ob<function>
418 }
419 Here, <unicode-codepoint> must be hexadecimal.
420 Before explaining each parameter, introduce some examples.
421 useenv itemize {
422   \item \builtin|mathchardef|\ob.opening\ob0\ob29d8\ob\verb|\lfoo|
423   \item \builtin|mathchardef|\ob.ordinary\ob0\ob2202\ob\verb|\diff|
424 }
425
426 \subsection{\builtin|get_filepath| builtin}
427 Because of the Vesti internal, if one includes picture by using
428 \verb|\includegraphics| for instance, providing the raw path cause an error from
429 \LaTeX{} with \lq\lq file not found\rq\rq. \builtin|get_filepath| builtin adjust
430 raw file path into the new relative one which \LaTeX{} knows. Here is the example
431 code.
432 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@] {
433 %#\includegraphics{\get_filepath{./foo.png}}
434 }
435 Internally, \builtin|get_filepath| changes file path into the relative one with
436 respect to {\tt.vesti-dummy}. For example, if {\tt foo.png}, {\tt foo.ves}
437 and {\tt.vesti-dummy} are located in the root directory, then inside of {\tt
438 foo.ves}, \builtin|get_filepath|{\tt(./foo.png)} will changed into
439 {\tt../foo.png} because for {\tt.vesti-dummy}, {\tt foo.png} is located outside
440 of it.
441
442 \subsection{\builtin|at_on| and \builtin|at_off| builtins}
443 Both builtins are same as \LaTeX{} functions \verb|\makeatletter| and
444 \verb|\makeatother| plus changes Vesti lexer such that \verb|@| character is
445 also can be a \LaTeX{} function name. For instance, without using \builtin|at_on|
446 command, below code does not compiles.
447 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|&] {
448 %#+keyword|defun& @foo (#1) {Hello, #1!}
449 }
450 This is because \keyword{|%-defun-%|} expects a valid \LaTeX{} function name but
451 the character \verb|@| is not a valid one except after the \verb|\makeatletter|
452 function. Since \keyword{|%-defun-%|} is a Vesti grammar, there is no way to tell
453 Vesti that \verb|@| is a right one although one uses \verb|\makeatletter|.
454 By using \builtin|at_on|, below code then compiles.
455 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|&] {
456 %##at_on
457 %#+keyword|defun& @foo (#1) {Hello, #1!}
458 %##at_off
459 }
460 One can not using \builtin|at_off|, but I strongly recommand to match
461 pairs.
462
463 \subsection{\builtin|ltx3_on| and \builtin|ltx3_off| builtins}
464 Those builtins are similar with \builtin|at_on| and \builtin|at_off|
465 pairs but for \LaTeX3.
466
467 \subsection{\builtin|textmode| and \builtin|mathmode| builtins}
468 Internally, vesti tracks so called {\tt text mode} and {\tt math mode}.
469 In the math mode, for instance, the token \verb|->| is changed into $->$ inside
470 of the math mode. The idea is that Vesti changes \verb|->| into \verb|\to|, and
471 so on. But when defining some function using \defun*, for example, one might
472 want to define
473 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-  

474 +keyword|useenv@ foo {->-}  

475 -%}
476 so that \verb|$foo$| makes $-><-$, one should change the mode. Here,

```

```

477 \builtin|mathmode| comes into the place. One should write instead that
478 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-}
479 +keyword|useenv@ foo {#mathmode{-><-}}}
480 -%}

481 \section{Lua API}
482 Vesti uses Lua in both building script and inline inside of Vesti code.
483 Below are functions which are baked in Vesti compiler.
484 useenv table [ht] {
485     \centering
486     #lu:
487     local content = read_all("../src/Lua.zig")
488
489     local pat = '%.name%s*=%s*([^"]+)'
490
491     local builtins = {}
492     for name, tok in content:gmatch(pat) do
493         builtins[#builtins + 1] = name
494     end
495     table.sort(builtins)
496
497     vesti.print([[\begin{tabular}{ccccc}]])
498
499     for i, kw in ipairs(builtins) do
500         local cell = string.format("\tt{\%s}", kw)
501         if (i % 4) == 0 then
502             vesti.print(cell .. [[\\]])
503         else
504             vesti.print(cell .. "&")
505         end
506     end
507
508     vesti.print([[\end{tabular}]])
509     :lu#[readAll]
510     \caption{Keywords in Vesti}
511 }
512

514 \newpage
515 \section{Source Code of This Document}
516 Below code was generated by inline lua.
517 useenv Verbatim [numbers=left, numbersep=5pt, frame=single] {
518 #lu:
519     local content = read_all("vesti_man.ves")
520     for line in content:gmatch("(^\r\n*)\r?\n?") do
521         vesti.print(line)
522     end
523 :lu#[readAll]
524 }
525
526

```