

Vesti Transpiler User Manual

Sungbae Jeong

February 16, 2026

Contents

1	Introduction	1
2	Structure of Vesti File	2
3	Keywords	2
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
4	Builtins	5
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>#enum</code> and <code>#enum_counter</code> builtins	6
4.6	<code>#get_filepath</code> builtin	7
4.7	<code>#at_on</code> and <code>#at_off</code> builtins	8
4.8	<code>#ltx3_on</code> and <code>#ltx3_off</code> builtins	8
4.9	<code>#textmode</code> and <code>#mathmode</code> builtins	8
5	Lua API	8
6	Source Code of This Document	9

1 Introduction

I used to create several documents using L^AT_EX (or plain T_EX but T_EX 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 L^AT_EX 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 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^AT_EX. 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^AT_EX. 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 thier 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^AT_EX. 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>* \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^AT_EX, 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

defun is the keyword which makes a L^AT_EX function. Internally, it uses **xparse** package which is automatically included from modern L^AT_EX2 ϵ kernel. The grammar of **defun** is the following.

defun \sqcup [<attr>]? \sqcup <command-name> \sqcup (<param-spec>)? \sqcup {<implementation>}

Here, `?` means that this is an optional.

First, let us see some examples for `defun`, and then explain the meaning of each fields.

```
1 \defun foo {Hello, World!}
```

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 sometimes desirable to use names that can not be reserved 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     #def        #enum
#enum_counter #eq          #get_filepath #label      #ltx3_off
#ltx3_on     #mathchardef  #mathmode    #noltx3   #picture
#raw_tex      #showfont    #textmode    #undef
```

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 L^AT_EX 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 sugar 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 `#enum` and `#enum_counter` builtins

`#enum` builtin is just an enumerate environment with minimal support of `enumitem` feature. For example,

```

1 #enum {
2     \item aaa
3     \item bbb
4     \item ccc
5 }
```

prints

1. aaa
2. bbb
3. ccc

`#enum` also can be nested like follows.

```

1 #enum {
2     \item aaa
3     \item #enum {
4         \item bbb
5         \item ccc
6     }
7     \item ddd
8 }
```

It prints

1. aaa
2. (a) bbb

(b) ccc

3. ddd

To customize a format of `\item` like `enumitem`, one can write like the following:

```
1 #enum (\Roman*.) {
2     \item aaa
3     \item #enum ({\roman**}) {
4         \item bbb
5         \item ccc
6     }
7     \item ddd
8 }
```

I. aaa

II. i* bbb
ii* ccc

III. ddd

As one can see, `#enum` changes the format with inside of () . In Vesti, single * changes into {<enum count>} (including braces), and consecutive ** changes into single * character. Thus, `\Roman*.` changes into `\Roman{enumi}.` and `{\roman**}` changes into `{\roman{enumii}}*` (because this part is inside of another `#enum`).

Beware that by the parting mechanism, `\roman***` changes into `\roman*{<enum count>}`, and `\roman{*}**` into `\roman{{<enum count>}}*`, which are invalid arguments for `\roman` L^AT_EX internal command.

`#enum_counter` builtin is a helper to get the name of the current enum counter. For example, if one wants to start a second enum from 3, one can write

```
1 #enum {
2     \item aaa
3     \item #enum {
4         \setcounter{\#enum_counter}{2}
5         \item bbb
6         \item ccc
7     }
8     \item ddd
9 }
```

and it prints

1. aaa

2. (c) bbb
(d) ccc

3. ddd

4.6 `#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^AT_EX with “file not found”. `#get_filepath` builtin adjust raw file path into the new relative one which L^AT_EX 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.

4.7 #at_on and #at_off builtins

Both builtins are same as L^AT_EX functions \makeatletter and \makeatother plus changes Vesti lexer such that @ character is also can be a L^AT_EX function name. For instance, without using #at_on command, below code does not compile.

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

This is because defun expects a valid L^AT_EX 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 use #at_off, but I strongly recommend to match pairs.

4.8 #ltx3_on and #ltx3_off builtins

Those builtins are similar with #at_on and #at_off pairs but for L^AT_EX3.

4.9 #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 → 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 →←, 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	getCurrentDir	getEngineType
getModule	joinpath	mkdir	parse
ping	print	setCurrentDir	unzip
vestiDummyDir			

Table 3: Lua builtin functions 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     amsmath,
8 }
9
10 \title{Vesti Transpiler User Manual}
11 \author{Sungbae Jeong}
12 #::
13 local function read_all(path)
14     local f, err = io.open(path, "rb")
15     assert(f, ("cannot open %s: %s"):format(path, err))
16     local data = f:read("*a")
17     f:close()
18     return data
19 end
20 ::#<readAll>
21
22 -- definition of \keyword command
23 defun [!] keyword (m) {{\tt\color{purple}#1}}
24 defun [!] builtin (v) {{\tt\color{blue!65!yellow}\##1}}
25 #::
26 local commands = { "useenv", "defun", "defenv" }
27 for _, command in ipairs(commands) do
28     local s = string.format(
29         "defun %s (s) {\\"IfBooleanTF{#!1}{\\keyword{%%!-%s-%%}}{\\keyword{%%!-%s-%%} }}",
30         command, command, command
31     )
32     vesti.print(vesti.parse(s))
33 end
34 ::#
35
36 #::
37 function begenv(add_bang, add_space)
38     local name = "begenv"
39     if add_bang then name = name .. "!" end
40     local begenv = "\\\ keyword{" .. name .. "}"
41     if add_space then begenv = begenv .. " " end
42     vesti.print(begenv, { nl = 0 })
43 end
44
45 function endenv(add_bang, add_space)
46     local name = "endenv"
47     if add_bang then name = name .. "!" end
48     local endenv = "\\\ keyword{" .. name .. "}"
49     if add_space then endenv = endenv .. " " end
50     vesti.print(endenv, { nl = 0 })
51 end
52 ::#*
53
54 -- space character used in the textbook.
55 #at_on
56 #chardef 2423 \@b
```

```

57 defun ob {\kern2pt{\tt\@b}\kern2pt}
58 #at _off
59
60 startdoc
61 \maketitle
62 \tableofcontents
63
64 \section{Introduction}
I used to create several documents using \LaTeX\ (or plain \TeX\ but \TeX\ is
66 quite cumbersome to write--especially when working with very complex tables or
67 inserting images). Its markdown-like syntax is also not comfortable to use. For
68 example, here is a simple \LaTeX\ document:
69
70 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%
71 % coprime is my custom class. See https://github.com/e0328eric/coprime.
72 \documentclass[tikz, geometry]{coprime}
73
74 \settitle{My First Document}{Sungbae Jeong}{}
75 \setgeometry{a4paper, margin = 2.5cm}
76
77 \begin{document}
78 \section{Foo}
Hello, World!
79 \begin{figure}[ht]
80     \centering
81     \begin{tikzpicture}
82         \draw (0,0) -- (1,1);
83     \end{tikzpicture}
84 \end{figure}
85
86 The code above is a figure using TikZ.
87
88 \end{document}
89 -%}
90
91 What annoys me most when using it is the \lq\verb@\begin@\rq\ and
92 \lq\verb@\end@\rq\ blocks. Is there a way to write something much simpler? This
93 question led me to start this project. Currently, the following code is
94 generated into the \LaTeX code above (except comments) using vesti:
95
96 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%
97 % coprime is my custom class. See https://github.com/e0328eric/coprime.
98 +keyword|docclass@ coprime (tikz, geometry)
99
100 \settitle{My First Document}{Sungbae Jeong}{}
101 \setgeometry{a4paper, margin = 2.5cm}
102
103 +keyword|startdoc@
104
105 \section{Foo}
Hello, World!
106 +keyword|useenv@ figure [ht] {
107     \centering
108     +keyword|useenv@ tikzpicture {
109         \draw (0,0) -- (1,1);
110     }
111 }
112
113 }
114
115 The code above is a figure using TikZ.
116 -%}

```

```

117
118 Everywhere in this paper, the symbol \ob\ denotes a \lq\lq space\rq\rq\ when
119 one use Vesti.
120
121 \section{Structure of Vesti File}
122 Vesti is similar as \LaTeX. Its structure consists with two parts: {\tt preamble} and
123 {\tt main}. Preamble is the place where \LaTeX\ documentclass, packages, and
124 several settings are located. Main body is where actual documentation is located.
125 Below figure is the simple Vesti documentation.
126
127 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-}
128 +keyword|docclass@ article (10pt)
129 +keyword|importpkg@ {
130     geometry (a4paper, margin=2.2cm)
131 }
132 +keyword|startdoc@
133 Hello, Vesti!
134 -%
135
136 We will see later, but the very difference with \LaTeX\ is that Vesti has its
137 own keywords (keywords are colored with purple). It makes the code readable and
138 it is easier and faster to write the document. The keyword startdoc splits
139 the preamble and the main part of the documentation similar with
140
141 -- Don't ask why I chose Q for catcode 0.
142 %#\{\tt\catcode`Q=0 Q\catcode`\\=12 \begin{Q}{documentQ}\} in \LaTeX.
143 However, Vesti does not have the analogous part of
144 %#\{\tt\catcode`Q=0 Q\catcode`\\=12 \end{Q}{documentQ}\},
145 because almost every \LaTeX\ document (99.999\% I'm sure) does not have any code
146 below %#\{\tt\catcode`Q=0 Q\catcode`\\=12 \end{Q}{documentQ}\}.
147 For this reason, Vesti automatically ends document when EOF (End Of File) is
148 found.
149
150 \section{Keywords}
151 Followings are reserved as keywords. In this document, every Vesti keyword has
152 the form like \keyword{this}.
153 useenv table [ht] {
154     \centering
155     #::
156     local content = read_all("../src/lexer/Token.zig")
157
158     -- Lua's built-in patterns don't support lookahead.
159     -- We capture both the keyword and the TokenType, then filter out 'deprecated'.
160     -- Pattern breakdown:
161     -- %.%{      => matches ".{"
162     -- %s*"(["+) => a quoted string -> capture 1
163     -- %s*,%s*TokenType%.([%w_]++) => TokenType.<Name> -> capture 2
164     local pat = "%.%{%"s*"\("["+)\")%"s*,%"s*TokenType%.([%"w_]++)"
165
166     local keywords = {}
167     for name, tok in content:gmatch(pat) do
168         if tok ~= "deprecated" then
169             keywords[#!keywords + 1] = name
170         end
171     end
172
173     table.sort(keywords)
174
175     vesti.print([[\begin{tabular}{cccc}]])
176

```

```

177     for i, kw in ipairs(keywords) do
178         local cell = string.format("\\\ keyword{\%s}", kw)
179         if (i % 4) == 0 then
180             vesti.print(cell .. "[[\\""))
181         else
182             vesti.print(cell .. "&")
183         end
184     end
185
186     vesti.print([[\end{tabular}]])
187     ::#[readAll]
188     \caption{Keywords in Vesti}
189 }
190
191 \subsection{\keyword{docclass} keyword}
192 Keyword \keyword{docclass} is an analogous of \verb|\documentclass| in \LaTeX.
193 If \keyword{docclass} is in the main paragraph, it acts just a normal word.
194 In other words, \keyword{docclass} actives only in the preamble.
195 The syntax of \keyword{docclass} is following:
196
197 useenv center {
198     \keyword{docclass}\ob<class name>\ob{\tt{}<arguments>{\tt{}}
199 }
200 Here, arguments are separated by commas and embraced by {\tt ()}. Here are some
201 examples.
202
203 \goodbreak
204 useenv itemize {
205     \item \keyword{docclass}\ob{\tt article}
206     \item \keyword{docclass}\ob{\tt article\ob(10pt)}
207     \item \keyword{docclass}\ob{\tt article\ob(10pt,\ob twocols)}
208     \item \keyword{docclass}\ob{\tt article\ob(10pt,twocols)}
209 }
210
211 \subsection{\keyword{importpkg} keywords}
212 Keyword \keyword{importpkg} is an analogous of \verb|\usepackage| in \LaTeX.
213 If \keyword{importpkg} is in the main paragraph, it acts just a normal word.
214 In other words, \keyword{importpkg} actives only in the preamble.
215
216 importpkg has two different syntax. First one is same as docclass.
217 useenv center {
218     \keyword{importpkg}\ob<pkg-name>\ob{\tt{}<arguments>{\tt{}}
219 }
220 Here, arguments are separated by commas and embraced by {\tt ()}.
221 In the practical case, one should include several packages with options.
222 importpkg also supports such case. We will look at an example instead of
223 giving rigorous grammar.
224 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%
225 +keyword|importpkg@ {
226     amsmath, amssymb, amsthm,
227     geometry (a4paper, margin=2.2cm),
228 }
229 -%}
230
231 \noindent As one can see, inside of \verb|{|}, several packages can be used
232 together with thier options.
233
234 \subsection{\keyword{startdoc} keyword}
235 Keyword \keyword{startdoc} tells to Vesti that the main document starts. In the
236 main document, you can also write \keyword{startdoc} in the main document. In

```

```

237 | that case, \keyword{startdoc} does nothing.
238 |
239 | \subsection{\useenv keyword}
240 | As the name implies, keyword \useenv is an analogous of \verb|\begin{...}| and
241 | \verb|\end{...}| pair in \LaTeX.
242 | The simplest \useenv is like this.
243 |
244 | \useenv figure [ht] {
245 |   \centering
246 |   \useenv tikzpicture {
247 |     \useenv scope {
248 |       \path (0,0) node {\vbox{
249 |         %#\hbox{\tt\useenv center \{}}
250 |         %#\hbox{\tt\obeyspaces Hello, World!}
251 |         %#\hbox{\tt\obeyspaces\}}
252 |       }};
253 |     }
254 |     \path (2.3,0) node {or};
255 |     \useenv scope [shift={(6,0)}]{
256 |       \path (0,0) node {\tt\useenv center \{ Hello, World! \}};
257 |     }
258 |   }
259 | }
260 |
261 | As you can see, {\tt\useenv center} is the part of \verb|\begin{center}|, and
262 | the single {\tt\}} is the part of \verb|\end{center}|. Since Vesti knows their
263 | pair, one can write a code with several environment, and each pair is properly
264 | matched. For instance, above example is written in Vesti like follows. Here,
265 | \verb|\useenv| just prints \useenv in that style.
266 |
267 | \useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%
268 | |+color|purple@\useenv@ figure [ht] {
269 |   \centering
270 |   |+color|purple@\useenv@ tikzpicture {
271 |     |+color|purple@\useenv@ scope {
272 |       \path (0,0) node {\vbox{
273 |         |+color|blue@%#@\hbox{\tt\useenv center \{}}
274 |         |+color|blue@%#@\hbox{\tt\obeyspaces Hello, World!}
275 |         |+color|blue@%#@\hbox{\tt\obeyspaces\}}
276 |       }};
277 |     }
278 |     \path (2.3,0) node {or};
279 |     |+color|purple@\useenv@ scope [shift={(6,0)}] {
280 |       \path (0,0) node {\tt\useenv center \{ Hello, World! \}};
281 |     }
282 |   }
283 | }
284 | -%
285 |
286 | Full syntax about \useenv is the following.
287 | \useenv center {
288 |   \useenv\ob<environment name>\ob<argument>*\ob{\tt\{} <body> {\tt\}}
289 | }
290 | where '*' means that the number of <argument> is zero or at least one, and
291 | $$
292 |   "<argument>" = \useenv cases {
293 |     "(<argument>)" & "mandatory arguments" \cr
294 |     "[<argument>]" & "optional arguments" \cr
295 |   }
296 | $$

```

```

297 For instance, below one is a valid Vesti code (environment {\tt foo} is
298 undefined in general). As one can see, spaces cannot exist in between <argument>s.
299 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-}
300 |+color|purple@useenv@ foo (asd)(fff)[\ames and \awdsa](askws)[\rrsaa] {
301     foobar
302 }
303 -%}

304

305 #def #begenv {#:begenv(false,true)::#*}
306 #def #begenv_arg {#:begenv(#1,#2)::#*}
307 #def #endenv {#:endenv(false,true)::#*}
308 #def #endenv_arg {#:endenv(#1,#2)::#*}

309

310 \subsection{#begenv and #endenv keywords}
311 As the name implies, both keywords #begenv and #endenv are
312 analogous of \verb|\begin{...}| and \verb|\end{...}| pair in \LaTeX,
313 respectively. Thus below code
314 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-}
315 |+color|purple@begenv@ center
316 asdsad
317 |+color|purple@endenv@
318 -%}
319 is exactly same as
320 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-}
321 +keyword@useenv@ center {
322     asdsad
323 }
324 -%}
325 Then why we need #begenv and #endenv if we already have \useenv*?
326 The only reason which both #begenv and #endenv_arg(true)(false) exist is
327 defining new environment. For instance, one defines a new environment like
328 following.
329 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@] {
330 %#+keyword|defenv@ foo {|+color|purple@begenv@ minipage(\textwidth)}{|+color|purple@endenv@}
331 }

332

333 \subsection{\defun keyword}
334 \defun is the keyword which makes a \LaTeX\ function. Internally, it uses {\tt
335 xparse} package which is automatically included from modern \LaTeX$2\epsilon$ kernel. The grammar of \defun is the following.
336

337 begenv center
338     \defun\ob[<attr>]? \ob<command-name>\ob(<param-spec>)? \ob\{<implementation>\}
339 endenv
340 Here, {\tt`?`} means that this is an optional.

341 First, let us see some examples for \defun*, and then explain the meaning of
342 each fields.

343

344 :::
345 local examples = {
346     "\defun foo {Hello, World!}"
347 }

348 for _, example in ipairs(examples) do
349     vesti.print(vesti.parse(
350         "begenv! Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]"
351     ))
352     vesti.print(example)
353     vesti.print(vesti.parse("endenv! Verbatim"))

```

```

357 end
358 ::#
359
360 \section{Builtins}
361 Vesti also has its own builtin functions, which are prefixed with \#.
362 One might wonder what distinguishes builtins from keywords. In fact, from the
363 compiler's internal perspective, there is no real difference. However, in actual
364 language usage, constantly typing the prefix can be somewhat tedious, especially
365 for functions that are commonly used.
366
367 From the perspective of language design --particularly in Vesti-- it is
368 sometimes desirable to use names that can not be reserved as keywords. For example,
369 Vesti provides a built-in function \builtin{label}, which will be explained
370 later. Since Vesti is a typewriting-oriented language, the word \lq\lq
371 label\rq\rq\ is often used in its ordinary sense rather than in its special
372 semantic meaning within the language.
373
374 Followings are reserved as builtin functions.
375
376 useenv table [ht] {
377     \centering
378     #::
379     local content = read_all("../src/lexer/Token.zig")
380
381     -- match .{ "here" }
382     local pat = "%.%{%" s* \"([^\"]+)\"%s*%}"
383
384     local builtins = {}
385     for name, tok in content:gmatch(pat) do
386         builtins[#!builtins + 1] = name
387     end
388     table.sort(builtins)
389
390     vesti.print([[\begin{tabular}{ccccc}]])
391
392     for i, kw in ipairs(builtins) do
393         local cell = string.format("\\"builtin@%s@", kw)
394         if (i % 5) == 0 then
395             vesti.print(cell .. [[\\]])
396         else
397             vesti.print(cell .. "&")
398         end
399     end
400
401     vesti.print([[\end{tabular}]])
402     ::#[readAll]
403     \caption{Builtins in Vesti}
404 }
405 Since {\tt fancyvrb} does not allow verbatim-like commands inside of
406 {\tt Verbatim} environment, builtins are not colored in example codes.
407
408 \subsection{\builtin{label} builtin}
409 For the compiling issue, if one label some environment, one should write like
410 follows using \LaTeX\ function \verb|\label|.
411 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=&|@]{%
412 &keyword|useenv@ theorem { \label{eq:1}
413     1 + 1 = 2.
414 }
415 -%}
416

```

```

417 However, such code is not natural. Thus \builtin|label| comes into the place.
418 One can label environments using \builtin|label| builtin like this.
419 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=&|@]{%-
420 #label(eq:1) &keyword|useenv@ theorem {
421     1 + 1 = 2.
422 }
423 -%
424 One can also write like the following, which I personally prefer.
425 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=&|@]{%-
426 #label(eq:1)
427 &keyword|useenv@ theorem {
428     1 + 1 = 2.
429 }
430 -%
431
432 \subsection{\builtin|eq| builtin}
433 \builtin|eq| is a abbreviation of \lq\keyword{%-useenv-%} equation\rq.
434 Actually, below codes are same.
435
436 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-
437 #label(eq:1)
438 +keyword|useenv@ equation {
439     \sum_{n=1}^{\infty} {1/n^2} = {\pi^2/6}.
440 }
441 -%
442 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-
443 #eq(eq:1) {
444     \sum_{n=1}^{\infty} {1/n^2} = {\pi^2/6}.
445 }
446 -%
447 Since Vesti is used in the mathematical documents mainly, small syntactic suger
448 is needed, so Vesti introduce \builtin|eq| builtin.
449
450 \subsection{\builtin|chardef| builtin}
451 \builtin|chardef| defines a text token using the unicode codepoint.
452 Here is the grammar of \builtin|chardef| builtin.
453 useenv center {
454     \builtin|chardef|\ob<unicode-codepoint>\ob<function>
455 }
456 Here, <unicode-codepoint> must be hexadecimal.
457
458 \subsection{\builtin|mathchardef| builtin}
459 \builtin|mathchardef| defines a math token using the unicode codepoint.
460 Here is the grammar of \builtin|mathchardef| builtin.
461 useenv center {
462     \builtin|mathchardef|\ob<kind>\ob<font-num>\ob<unicode-codepoint>\ob<function>
463 }
464 Here, <unicode-codepoint> must be hexadecimal.
465 Before explaining each parameter, introduce some examples.
466 useenv itemize {
467     \item \builtin|mathchardef|\ob.opening\ob0\ob29d8\ob\verb|\lfooo|
468     \item \builtin|mathchardef|\ob.ordinary\ob0\ob2202\ob\verb|\verb|\diff|
469 }
470
471 \subsection{\builtin|enum| and \builtin|enum_counter| builtins}
472 \builtin|enum| builtin is just an enumerate environment with minimal support of
473 {\tt enumitem} feature. For example,
474 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-
475 #enum {
476     \item aaa

```

```

477     \item bbb
478     \item ccc
479 }
480 -%
481 prints
482 #enum {
483     \item aaa
484     \item bbb
485     \item ccc
486 }
487 \builtin|enum| also can be nested like follows.
488 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-}
489 #enum {
490     \item aaa
491     \item #enum {
492         \item bbb
493         \item ccc
494     }
495     \item ddd
496 }
497 -%
498 It prints
499 #enum {
500     \item aaa
501     \item #enum {
502         \item bbb
503         \item ccc
504     }
505     \item ddd
506 }
507 To customize a format of \verb|\item| like {\tt enumitem}, one can
508 write like the following:
509 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-}
510 #enum (\Roman*.) {
511     \item aaa
512     \item #enum ({\roman}**)
513         \item bbb
514         \item ccc
515     }
516     \item ddd
517 }
518 -%
519 #enum (\Roman*.) {
520     \item aaa
521     \item #enum ({\roman}**)
522         \item bbb
523         \item ccc
524     }
525     \item ddd
526 }
527 As one can see, \verb|\enum| changes the format with inside of {\tt ()}.
528 In Vesti, single {\tt *} changes into {\tt \{<enum count>\}} (including braces),
529 and consecutive {\tt **} changes into single {\tt *} character.
530 Thus, \verb|\Roman*.| changes into \verb|\Roman{enumi}.| and \verb|\{\roman}**|
531 changes into \verb|\{\roman{enumii}\}*| (because this part is inside of another
532 \verb|\enum|).
533 Beware that by the parting mechanism, \verb|\roman***| changes into
534 \verb|\roman*{<enum count>}|, and \verb|\roman{*}**| into \verb|\roman{{<enum count>}}*|,
535 which are invalid arguments for \verb|\roman| \LaTeX internal command.

```

```

537 \builtin|enum_counter| builtin is a helper to get the name of the current enum
538 counter. For example, if one wants to start a second enum from 3, one can write
539 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-}
540 #enum {
541     \item aaa
542     \item #enum {
543         \setcounter{#enum_counter}{2}
544         \item bbb
545         \item ccc
546     }
547     \item ddd
548 }
549 -%}
550 and it prints
551 #enum {
552     \item aaa
553     \item #enum {
554         \setcounter{#enum_counter}{2}
555         \item bbb
556         \item ccc
557     }
558     \item ddd
559 }
560 }

561 \subsection{\builtin|get_filepath| builtin}
562 Because of the Vesti internal, if one includes picture by using
563 \verb|\includegraphics| for instance, providing the raw path cause an error from
564 \LaTeX\ with \lq\lq file not found\rq\rq. \builtin|get_filepath| builtin adjust
565 raw file path into the new relative one which \LaTeX\ knows. Here is the example
566 code.
567 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@] {
568 %#\includegraphics{#get_filepath{./foo.png}}
569 }
570 Internally, \builtin|get_filepath| changes file path into the relative one with
571 respect to {\tt.vesti-dummy}. For example, if {\tt foo.png}, {\tt foo.ves}
572 and {\tt.vesti-dummy} are located in the root directory, then inside of {\tt
573 foo.ves}, \builtin|get_filepath|{\tt(./foo.png)} will changed into
574 {\tt../foo.png} because for {\tt.vesti-dummy}, {\tt foo.png} is located outside
575 of it.

576 \subsection{\builtin|at_on| and \builtin|at_off| builtins}
577 Both builtins are same as \LaTeX\ functions \verb|\makeatletter| and
578 \verb|\makeatother| plus changes Vesti lexer such that \verb|@| character is
579 also can be a \LaTeX\ function name. For instance, without using \builtin|at_on|
580 command, below code does not compiles.
581 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|&] {
582 %#+keyword|defun& @foo (#1) {Hello, #1!}
583 }
584 This is because \verb|keyword{%-defun-%}| expects a valid \LaTeX\ function name but
585 the character \verb|@| is not a valid one except after the \verb|\makeatletter|
586 function. Since \verb|keyword{%-defun-%}| is a Vesti grammar, there is no way to tell
587 Vesti that \verb|@| is a right one although one uses \verb|\makeatletter|.
588 By using \builtin|at_on|, below code then compiles.
589 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|&]{%-}
590 #at_on
591 +keyword|defun& @foo (#1) {Hello, #1!}
592 #at_off
593 -%}
594 One can not using \builtin|at_off|, but I strongly recommand to match

```

```

597 pairs.
598
599 \subsection{\builtin{ltx3_on} and \builtin{ltx3_off} builtins}
600 Those builtins are similar with \builtin{at_on} and \builtin{at_off}
601 pairs but for \LaTeX3.
602
603 \subsection{\builtin{textmode} and \builtin{mathmode} builtins}
604 Internally, vesti tracks so called {\tt text mode} and {\tt math mode}.
605 In the math mode, for instance, the token \verb|->| is changed into $->$ inside
606 of the math mode. The idea is that Vesti changes \verb|->| into \verb|\to|, and
607 so on. But when defining some function using \defun*, for example, one might
608 want to define
609 \useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%
610 +keyword|useenv@ foo {->-}
611 -%}
612 so that \verb|$-><-$ makes $-><-$, one should change the mode. Here,
613 \builtin{mathmode} comes into the place. One should write instead that
614 \useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%
615 +keyword|useenv@ foo {#mathmode{->-}}
616 -%}
617
618 \section{Lua API}
619 Vesti uses Lua in both building script and inline inside of Vesti code.
620 Below are functions which are baked in Vesti compiler.
621 \useenv table [ht] {
622     \centering
623     #::
624     local content = read_all("../src/Lua.zig")
625
626     local pat = '%.name%s*=%s*([^\n]+)'
627
628     local lua = {}
629     for name, tok in content:gmatch(pat) do
630         lua[#!lua + 1] = name
631     end
632     table.sort(lua)
633
634     vesti.print([[\begin{tabular}{ccccc}]])
635
636     for i, kw in ipairs(lua) do
637         local cell = string.format("\\"tt">%s", kw)
638         if (i % 4) == 0 then
639             vesti.print(cell .. [[\\]])
640         else
641             vesti.print(cell .. "&")
642         end
643     end
644
645     vesti.print([[\end{tabular}]])
646     ::#[readAll]
647     \caption{Lua builtin functions in Vesti}
648 }
649
650
651 \newpage
652 \section{Source Code of This Document}
653 Below code was generated by inline lua.
654 \useenv Verbatim [numbers=left, numbersep=5pt, frame=single] {
655 #::
656     local content = read_all("vesti_man.ves")

```

```
657     for line in content:gmatch("([^\r\n]*)\r?\n?") do
658         vesti.print(line)
659     end
660 ::#[readAll]
661 }
662
```