

Vesti Transpiler User Manual

Sungbae Jeong

December 8, 2025

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 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^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 –particularly in Vesti– it is 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     #enum        #enum_counter
#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 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
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     amsmath,
8 }
9
10 \title{Vesti Transpiler User Manual}
11 \author{Sungbae Jeong}
12 #:readAll#
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:#<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 #:1#
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 #:1#
35
36 #:begenv#
37 function begenv(add_space, add_bang)
38     local name = "begenv"
39     if add_bang then name = name .. "!" end
40     if add_space then name = name .. " " end
41     vesti.print("\\keyword{" .. name .. "}")
42 end
43
44 function endenv(add_space, add_bang)
45     local name = "endenv"
46     if add_bang then name = name .. "!" end
47     if add_space then name = name .. " " end
48     vesti.print("\\keyword{" .. name .. "}")
49 end
50 #:begenv:#*
51
52 % space character used in the texbook.
53 #at_on
54 #chardef 2423 \@b
55 defun ob {\kern2pt{\tt\@b}\kern2pt}
56 #at_off
```

```

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

```

```

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

```

```

177     if (i % 4) == 0 then
178         vesti.print(cell .. [[\\]])
179     else
180         vesti.print(cell .. "&")
181     end
182 end
183
184 vesti.print([[\\end{tabular}]])
185 #keywords:#[readAll]
186 \caption{Keywords in Vesti}
187 }
188
189 \subsection{\keyword{docclass} keyword}
190 Keyword \keyword{docclass} is an analogous of \verb|\documentclass| in \LaTeX.
191 If \keyword{docclass} is in the main paragraph, it acts just a normal word.
192 In other words, \keyword{docclass} actives only in the preamble.
193 The syntax of \keyword{docclass} is following:
194
195 useenv center {
196     \keyword{docclass}\ob<class name>\ob{\tt{}<arguments>{\tt{}}
197 }
198 Here, arguments are separated by commas and embraced by {\tt ()}. Here are some
199 examples.
200
201 \goodbreak
202 useenv itemize {
203     \item \keyword{docclass}\ob{\tt article}
204     \item \keyword{docclass}\ob{\tt article\ob(10pt)}
205     \item \keyword{docclass}\ob{\tt article\ob(10pt,\ob twocols)}
206     \item \keyword{docclass}\ob{\tt article\ob(10pt,twocols)}
207 }
208
209 \subsection{\keyword{importpkg} keywords}
210 Keyword \keyword{importpkg} is an analogous of \verb|\usepackage| in \LaTeX.
211 If \keyword{importpkg} is in the main paragraph, it acts just a normal word.
212 In other words, \keyword{importpkg} actives only in the preamble.
213
214 importpkg has two different syntax. First one is same as docclass.
215 useenv center {
216     \keyword{importpkg}\ob<pkg-name>\ob{\tt{}<arguments>{\tt{}}
217 }
218 Here, arguments are separated by commas and embraced by {\tt ()}.
219 In the practical case, one should include several packages with options.
220 importpkg also supports such case. We will look at an example instead of
221 giving rigorous grammar.
222 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%
223 +\keyword{importpkg} @ {
224     amsmath, amssymb, amsthm,
225     geometry (a4paper, margin=2.2cm),
226 }
227 -%}
228
229 \noindent As one can see, inside of \verb|{|}|, several packages can be used
230 together with thier options.
231
232 \subsection{\keyword{startdoc} keyword}
233 Keyword \keyword{startdoc} tells to Vesti that the main document starts. In the
234 main document, you can also write \keyword{startdoc} in the main document. In
235 that case, \keyword{startdoc} does nothing.
236

```

```

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

```

```

297 |useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-  

298 |+color|purple@useenv@ foo (asd)(fff)[\ames and \awdsa](askws)[\rrsaa] {  

299   foobar  

300 }  

301 -%}  

302  

303 \subsection{#::#begenv()#::#* and #::#endenv()#::#* keywords}  

304 As the name implies, both keywords #::#begenv()#::#* and #::#endenv()#::#* are  

305 analogous of \verb|\begin{...}| and \verb|\end{...}| pair in \LaTeX,  

306 respectively. Thus below code  

307 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-  

308 |+color|purple@begenv@ center  

309   asdsad  

310 |+color|purple@endenv@  

311 -%}  

312 is exactly same as  

313 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-  

314 +keyword|useenv@ center {  

315   asdsad  

316 }  

317 -%}  

318 Then why we need #::#begenv()#::#* and #::#endenv()#::#* if we already have \useenv*?  

319 The only reason which both #::#begenv()#::#* and #::#endenv()#::#* exist is  

320 defining new environment. For instance, one defines a new environment like  

321 following.  

322 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@] {  

323 %#+keyword|defenv@ foo {|+color|purple@begenv@ minipage(\textwidth)}{|+color|purple@endenv@}  

324 }  

325  

326 \subsection{\defun keyword}  

327 \defun is the keyword which makes a \LaTeX\ function. Internally, it uses {\tt  

328 xpars} package which is automatically included from modern \LaTeX$2\epsilon$  

329 kernel. The grammar of \defun is the following.  

330  

331 begenv center  

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

333 endenv  

334 Here, {\tt`?`} means that this is an optional.  

335  

336 First, let us see some examples for \defun*, and then explain the meaning of  

337 each fields.  

338  

339 #:examples#  

340 local examples = {  

341   "\\\defun foo {Hello, World!}"  

342 }  

343  

344 for _, example in ipairs(examples) do  

345   vesti.print(vesti.parse(  

346     "begenv! Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]"  

347   ))  

348   vesti.print(example)  

349   vesti.print(vesti.parse("endenv! Verbatim"))  

350 end  

351 #examples:#  

352  

353 \section{Builtins}  

354 Vesti also has its own builtin functions, which are prefixed with \#.   

355 One might wonder what distinguishes builtins from keywords. In fact, from the  

356 compiler's internal perspective, there is no real difference. However, in actual

```

```

357 language usage, constantly typing the prefix can be somewhat tedious, especially
358 for functions that are commonly used.
359
360 From the perspective of language design --particularly in Vesti-- it is
361 sometimes desirable to use names that can not be reserved as keywords. For example,
362 Vesti provides a built-in function \builtin{label}, which will be explained
363 later. Since Vesti is a typewriting-oriented language, the word \lq\lq
364 label\rq\rq\ is often used in its ordinary sense rather than in its special
365 semantic meaning within the language.
366
367 Followings are reserved as builtin functions.
368
369 useenv table [ht] {
370   \centering
371   #:builtins#
372   local content = read_all("../src/lexer/Token.zig")
373
374   -- match .{ "here" }
375   local pat = "%.%{%"s*\\"([^\"]+)\\"%s*%}"
376
377   local builtins = {}
378   for name, tok in content:gmatch(pat) do
379     builtins[#builtins + 1] = name
380   end
381   table.sort(builtins)
382
383   vesti.print([[\begin{tabular}{ccccc}]])
384
385   for i, kw in ipairs(builtins) do
386     local cell = string.format("\\"builtin@%s@", kw)
387     if (i % 5) == 0 then
388       vesti.print(cell .. [[\\]])
389     else
390       vesti.print(cell .. "&")
391     end
392   end
393
394   vesti.print([[\end{tabular}]])
395   #builtins:#[readAll]
396   \caption{Builtins in Vesti}
397 }
398 Since {\tt fancyvrb} does not allow verbatim-like commands inside of
399 {\tt Verbatim} environment, builtins are not colored in example codes.
400
401 \subsection{\builtin{label} builtin}
402 For the compiling issue, if one label some environment, one should write like
403 follows using \LaTeX\ function \verb|\label|.
404 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=&|@]{%
405 &keyword|useenv@ theorem { \label{eq:1}
406   1 + 1 = 2.
407 }
408 -%}
409
410 However, such code is not natural. Thus \builtin{label} comes into the place.
411 One can label environments using \builtin{label} builtin like this.
412 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=&|@]{%
413 #label(eq:1) &keyword|useenv@ theorem {
414   1 + 1 = 2.
415 }
416 -%}

```

```

417 One can also write like the following, which I personally prefer.
418 \useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=&|@]{%-}
419 #label(eq:1)
420 &keyword|useenv@ theorem {
421     1 + 1 = 2.
422 }
423 -%}
424
425 \subsection{\builtin|eq| builtin}
426 \builtin|eq| is a abbreviation of \lq\keyword{%-useenv-%} equation\rq.
427 Actually, below codes are same.
428
429 \useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-}
430 #label(eq:1)
431 +keyword|useenv@ equation {
432     \sum_{n=1}^{\infty} {1/n^2} = {\pi^2/6}.
433 }
434 -%}
435 \useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-}
436 #eq(eq:1) {
437     \sum_{n=1}^{\infty} {1/n^2} = {\pi^2/6}.
438 }
439 -%}
440 Since Vesti is used in the mathematical documents mainly, small syntactic suger
441 is needed, so Vesti introduce \builtin|eq| builtin.
442
443 \subsection{\builtin|chardef| builtin}
444 \builtin|chardef| defines a text token using the unicode codepoint.
445 Here is the grammar of \builtin|chardef| builtin.
446 \useenv center {
447     \builtin|chardef|\ob<unicode-codepoint>\ob<function>
448 }
449 Here, <unicode-codepoint> must be hexadecimal.
450
451 \subsection{\builtin|mathchardef| builtin}
452 \builtin|mathchardef| defines a math token using the unicode codepoint.
453 Here is the grammar of \builtin|mathchardef| builtin.
454 \useenv center {
455     \builtin|mathchardef|\ob<kind>\ob<font-num>\ob<unicode-codepoint>\ob<function>
456 }
457 Here, <unicode-codepoint> must be hexadecimal.
458 Before explaining each parameter, introduce some examples.
459 \useenv itemize {
460     \item \builtin|mathchardef|\ob.opening\ob0\ob29d8\ob\verb|\lfooo|
461     \item \builtin|mathchardef|\ob.ordinary\ob0\ob2202\ob\verb|\verb|\diff|
462 }
463
464 \subsection{\builtin|enum| and \builtin|enum_counter| builtins}
465 \builtin|enum| builtin is just an enumerate environment with minimal support of
466 {\tt enumitem} feature. For example,
467 \useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-}
468 #enum {
469     \item aaa
470     \item bbb
471     \item ccc
472 }
473 -%}
474 prints
475 #enum {
476     \item aaa

```

```

477     \item bbb
478     \item ccc
479 }
480 \builtin{enum} also can be nested like follows.
481 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%
482 #enum {
483     \item aaa
484     \item #enum {
485         \item bbb
486         \item ccc
487     }
488     \item ddd
489 }
490 -%
491 It prints
492 #enum {
493     \item aaa
494     \item #enum {
495         \item bbb
496         \item ccc
497     }
498     \item ddd
499 }
500 To customize a format of \verb|\item| like {\tt enumitem}, one can
501 write like the following:
502 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%
503 #enum (\Roman*.) {
504     \item aaa
505     \item #enum ({\roman**} {
506         \item bbb
507         \item ccc
508     }
509     \item ddd
510 }
511 -%
512 #enum (\Roman*.) {
513     \item aaa
514     \item #enum ({\roman**} {
515         \item bbb
516         \item ccc
517     }
518     \item ddd
519 }
520 As one can see, \builtin{enum} changes the format with inside of {\tt()}.
521 In Vesti, single {\tt *} changes into {\tt \{\<enum count>\}} (including braces),
522 and consecutive {\tt **} changes into single {\tt *} character.
523 Thus, \verb|\Roman.| changes into \verb|\Roman{enumi}.| and \verb|\{\roman**}|
524 changes into \verb|\{\roman{enumii}\}*| (because this part is inside of another
525 \builtin{enum}).
526
527 Beware that by the parting mechanism, \verb|\roman***| changes into
528 \verb|\roman*<enum count>|, and \verb|\roman{*}**| into \verb|\roman{{<enum count>}}*|,
529 which are invalid arguments for \verb|\roman| \LaTeX internal command.
530
531 \builtin{enum_counter} builtin is a helper to get the name of the current enum
532 counter. For example, if one wants to start a second enum from 3, one can write
533 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%
534 #enum {
535     \item aaa
536     \item #enum {

```

```

537     \setcounter{#enum_counter}{2}
538     \item bbb
539     \item ccc
540   }
541   \item ddd
542 }
543 -%
544 and it prints
545 #enum {
546   \item aaa
547   \item #enum {
548     \setcounter{#enum_counter}{2}
549     \item bbb
550     \item ccc
551   }
552   \item ddd
553 }

554
555 \subsection{\builtin{get_filepath} builtin}
556 Because of the Vesti internal, if one includes picture by using
557 \verb|\includegraphics| for instance, providing the raw path cause an error from
558 \LaTeX{} with \lq\lq file not found\rq\rq. \builtin{get_filepath} builtin adjust
559 raw file path into the new relative one which \LaTeX{} knows. Here is the example
560 code.
561 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@] {
562 %#\includegraphics{\get_filepath{./foo.png}}
563 }
564 Internally, \builtin{get_filepath} changes file path into the relative one with
565 respect to {\tt.vesti-dummy}. For example, if {\tt foo.png}, {\tt foo.ves}
566 and {\tt.vesti-dummy} are located in the root directory, then inside of {\tt
567 foo.ves}, \builtin{get_filepath}{\tt{./foo.png}} will changed into
568 {\tt{./foo.png}} because for {\tt.vesti-dummy}, {\tt foo.png} is located outside
569 of it.
570
571 \subsection{\builtin{at_on} and \builtin{at_off} builtins}
572 Both builtins are same as \LaTeX{} functions \verb|\makeatletter| and
573 \verb|\makeatother| plus changes Vesti lexer such that \verb|@| character is
574 also can be a \LaTeX{} function name. For instance, without using \builtin{at_on}
575 command, below code does not compiles.
576 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|&] {
577 %#+keyword|defun& @foo (#1) {Hello, #1!}
578 }
579 This is because \verb|keyword{%-defun-%}| expects a valid \LaTeX{} function name but
580 the character \verb|@| is not a valid one except after the \verb|\makeatletter|
581 function. Since \verb|keyword{%-defun-%}| is a Vesti grammar, there is no way to tell
582 Vesti that \verb|@| is a right one although one uses \verb|\makeatletter|.
583 By using \builtin{at_on}, below code then compiles.
584 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|&]{%-}
585 #at_on
586 +keyword|defun& @foo (#1) {Hello, #1!}
587 #at_off
588 -%
589 One can not using \builtin{at_off}, but I strongly recommand to match
590 pairs.
591
592 \subsection{\builtin{ltx3_on} and \builtin{ltx3_off} builtins}
593 Those builtins are similar with \builtin{at_on} and \builtin{at_off}
594 pairs but for \LaTeX3.
595
596 \subsection{\builtin{textmode} and \builtin{mathmode} builtins}

```

```

597 Internally, vesti tracks so called {\tt text mode} and {\tt math mode}.
598 In the math mode, for instance, the token \verb|->| is changed into $->$ inside
599 of the math mode. The idea is that Vesti changes \verb|->| into \verb|\verb|\to|, and
600 so on. But when defining some function using \defun*, for example, one might
601 want to define
602 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-  

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

604 -%}
605 so that \verb|$-\>-$| makes $->-$, one should change the mode. Here,
606 \builtin{mathmode} comes into the place. One should write instead that
607 useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@]{%-  

608 +keyword|useenv@ foo {#mathmode{->-}}  

609 -%}
610
611 \section{Lua API}
612 Vesti uses Lua in both building script and inline inside of Vesti code.
613 Below are functions which are baked in Vesti compiler.
614 useenv table [ht] {
615     \centering
616     #:lua#
617     local content = read_all("../src/Lua.zig")
618
619     local pat = '%.name%s*=%s*([^\"]+)'
620
621     local builtins = {}
622     for name, tok in content:gmatch(pat) do
623         builtins[#builtins + 1] = name
624     end
625     table.sort(builtins)
626
627     vesti.print([[\begin{tabular}{ccccc}]])
628
629     for i, kw in ipairs(builtins) do
630         local cell = string.format("\tt{%s}", kw)
631         if (i % 4) == 0 then
632             vesti.print(cell .. [[\\]])
633         else
634             vesti.print(cell .. "&")
635         end
636     end
637
638     vesti.print([[\end{tabular}]])
639     #:lua:#[readAll]
640     \caption{Keywords in Vesti}
641 }
642
643
644 \newpage
645 \section{Source Code of This Document}
646 Below code was generated by inline lua.
647 useenv Verbatim [numbers=left, numbersep=5pt, frame=single] {
648 #:verbatim#
649     local content = read_all("vesti_man.ves")
650     for line in content:gmatch("(^\r\n*)\r?\n?") do
651         vesti.print(line)
652     end
653     #:verbatim:#[readAll]
654 }

```