Vesti Transpiler User Manual

Sungbae Jeong

October 29, 2025

Contents

1 Introduction

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.

```
docclass article (10pt)
importpkg {
    geometry (a4paper, margin=2.2cm)
}
startdoc
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 LaTeX. 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` <class name> (<arguments>)

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

- `docclass article`
- `docclass article (10pt)`
- `docclass article (10pt, twocols)`
- `docclass article (10pt,twocols)`

3.2 `importpkg` keywords

Keyword `importpkg` is an analogous of `\usepackage` in LaTeX. 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` pkg-name (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.

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

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 LaTeX. 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.

```
useenv figure [ht] {
    \centering
    useenv tikzpicture {
        useenv scope {
            \path (0,0) node {\vbox{
            %#\hbox{\tt\useenv center \{}
            %#\hbox{\tt\obeyspaces    Hello, World!}
            %#\hbox{\tt\obeyspaces\}}
            }};
        }
        \path (2.3,0) node {or};
        useenv scope [shift={(6,0)}] {
            \path (0,0) node {\tt\useenv center \{ Hello, World! \}};
        }
    }
}
```

Full syntax about useenv is the following.

useenv <environment name> <argument>* { <body> }

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

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

For instance, below one is a valid Vesti code (environment `foo` is undefined in general). As one can see, spaces can exist in between <argument>s.

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

3.5 begenv keyword and endenv keyword

As the name implies, both keywords begenv and endenv are analogous of `\begin{...}` and `\end{...}` pair in LaTeX, respectively. Thus below code

```
begenv center
    asdsad
endenv center
```

is exactly same as

```
useenv center {
    asdsad
}
```

Then why we need begenv and endenv if we already have useenv?

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.

#chardef	#enum	#eq	#get_filepath	#label
#ltx3_off	#ltx3_on	#makeatletter	#makeatother	#mathchardef
#mathmode	#noltx3	#nonstopmode	#picture	#showfont
#textmode				

Table 2: Builtins in Vesti

5 Source Code of This Document

Below code was generated by inline lua.

```
docclass article (10pt)
importpkg {
    geometry (a4paper, margin = 2.2cm),
    xcolor,
    tikz,
    fancyvrb,
}

\title{Vesti Transpiler User Manual}
\author{Sungbae Jeong}

importves (font.ves)

% read file contents using lua
#lu:
local function read_all(path)
  local f, err = io.open(path, "rb")
  assert(f, ("cannot open %s: %s"):format(path, err))
  local data = f:read("*a")
  f:close()
  return data
end
:lu#<readAll>

% definition of \keyword command
defun [!] keyword (m) {{\tt\color{purple}#1}}
defun useenv (s) {\IfBooleanTF{#1}{\keyword{%-useenv-%}}{\keyword{%-useenv-%} }}
defun begenv (s) {\IfBooleanTF{#1}{\keyword{%-begenv-%}}{\keyword{%-begenv-%} }}
defun endenv (s) {\IfBooleanTF{#1}{\keyword{%-endenv-%}}{\keyword{%-endenv-%} }}

startdoc
\maketitle
\tableofcontents

\section{Introduction}

\section{Structure of Vesti File}
Vesti is similar as \LaTeX. Its structure consists with two parts: {\tt preamble} and
{\tt 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.

useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@] {
%#|+color|purple@docclass@ article (10pt)
%#|+color|purple@importpkg@ {
%#    geometry (a4paper, margin=2.2cm)
%#}
%#|+color|purple@startdoc@
%#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

% Don't ask why I chose Q for catcode 0.
%#{\tt\catcode`Q=0 Qcatcode`\\=12 \beginQ{documentQ}} in \LaTeX.
However, Vesti does not have the analogous part of
%#{\tt\catcode`Q=0 Qcatcode`\\=12 \endQ{documentQ}},
because almost every \LaTeX\ document (99.999\% I'm sure) does not have any code
below %#{\tt\catcode`Q=0 Qcatcode`\\=12 \endQ{documentQ}}.
For this reason, Vesti automatically ends document when EOF (End Of File) is
found.

\section{Keywords}
Followings are reserved as keywords. In this document, every Vesti keyword has
the form like \keyword{this}.
useenv table [ht] {
    \centering
    #lu:
    local content = read_all("../src/lexer/Token.zig")

    -- Lua’s built-in patterns don’t support lookahead.
    -- We capture both the keyword and the TokenType, then filter out 'deprecated'.
    -- Pattern breakdown:
    --   %.%{         => matches ".{"
    --   %s*"([^"]+)" => a quoted string -> capture 1
    --   %s*,%s*TokenType%.([%w_]+) => TokenType.<Name> -> capture 2
    local pat = "%.%{%s*\"([^\"]+)\"%s*,%s*TokenType%.([%w_]+)"

    local keywords = {}
    for name, tok in content:gmatch(pat) do
      if tok ~= "deprecated" then
        keywords[#keywords + 1] = name
      end
    end

    table.sort(keywords)

    vesti.print([[\begin{tabular}{cccc}]])

    for i, kw in ipairs(keywords) do
        local cell = string.format("\\keyword{%s}", kw)
        if (i % 4) == 0 then
            vesti.print(cell .. [[\\]])
        else
            vesti.print(cell .. "&")
        end
    end

    vesti.print([[\end{tabular}]])
    :lu#[readAll]
    \caption{Keywords in Vesti}
}

\subsection{\keyword{docclass} keyword}
Keyword \keyword{docclass} is an analogous of \verb|\documentclass| in \LaTeX.
If \keyword{docclass} is in the main paragraph, it acts just a normal word.
```

```
In other words, \keyword{docclass} actives only in the preamble.
The syntax of \keyword{docclass} is following:

useenv center {
    \keyword{docclass}\kern0.5em <class name>\kern0.5em {\tt(}<arguments>{\tt)}
}
Here, arguments are separated by commas and embraced by {\tt ()}. Here are some
examples.

\goodbreak
useenv itemize {
    \item \keyword{docclass} {\tt article}
    \item \keyword{docclass} {\tt article (10pt)}
    \item \keyword{docclass} {\tt article (10pt, twocols)}
    \item \keyword{docclass} {\tt article (10pt,twocols)}
}

\subsection{\keyword{importpkg} keywords}
Keyword \keyword{importpkg} is an analogous of \verb|\usepackage| in \LaTeX.
If \keyword{importpkg} is in the main paragraph, it acts just a normal word.
In other words, \keyword{importpkg} actives only in the preamble.

importpkg has two different syntax. First one is same as docclass.
useenv center {
    \keyword{importpkg}\kern1em pkg-name\kern1em {\tt(}arguments{\tt)}
}
Here, arguments are separated by commas and embraced by {\tt ()}.
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.
useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@] {
%#|+color|purple@importpkg@ {
%#    amsmath, amssymb, amsthm,
%#    geometry (a4paper, margin=2.2cm),
%#}
}

\noindent As one can see, inside of \verb|{}|, several packages can be used
together with thier options.

\subsection{\keyword{startdoc} keyword}
Keyword \keyword{startdoc} tells to Vesti that the main document starts. In the
main document, you can also write \keyword{startdoc} in the main document. In
that case, \keyword{startdoc} does nothing.

\subsection{\useenv keyword}
As the name implies, keyword \useenv is an analogous of \verb|\begin{...}| and
\verb|\end{...}| pair in \LaTeX.
The simplest \useenv is like this.

useenv figure [ht] {
    \centering
    useenv tikzpicture {
        useenv scope {
            \path (0,0) node {\vbox{
            %#\hbox{\tt\useenv center \{}
            %#\hbox{\tt\obeyspaces    Hello, World!}
            %#\hbox{\tt\obeyspaces\}}
            }};
        }
        \path (2.3,0) node {or};
        useenv scope [shift={(6,0)}]{
            \path (0,0) node {\tt\useenv center \{ Hello, World! \}};
        }
    }
}

As you can see, {\tt\useenv center} is the part of \verb|\begin{center}|, and
the single {\tt\}} is the part of \verb|\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,
\verb|\useenv| just prints \useenv in that style.

useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@] {
%#|+color|purple@useenv@ figure [ht] {
%#    \centering
%#    |+color|purple@useenv@ tikzpicture {
%#        |+color|purple@useenv@ scope {
%#            \path (0,0) node {\vbox{
%#            |+color|blue@%#@\hbox{\tt\useenv center \{}
%#            |+color|blue@%#@\hbox{\tt\obeyspaces    Hello, World!}
%#            |+color|blue@%#@\hbox{\tt\obeyspaces\}}
%#            }};
%#        }
%#        \path (2.3,0) node {or};
%#        |+color|purple@useenv@ scope [shift={(6,0)}] {
%#            \path (0,0) node {\tt\useenv center \{ Hello, World! \}};
%#        }
%#    }
%#}
}

Full syntax about \useenv is the following.
useenv center {
    \useenv\kern0.5em <environment name>\kern0.5em <argument>*\kern0.5em
    {\tt\{} <body> {\tt\}}
}
where `*' means that the number of <argument> is zero or at least one, and
$$
    "<argument>" = useenv cases {
        "(<argument>)" & "mandatory arguments" \\
        "[<argument>]" & "optional arguments"
    }
$$
For instance, below one is a valid Vesti code (environment {\tt foo} is
undefined in general). As one can see, spaces can exist in between <argument>s.
useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@] {
%#|+color|purple@useenv@ foo (asd)(fff)[\ames and \awdsa] (askws) [\rrsaa] {
%#    foobar
%#}
}

\subsection{\begenv keyword and \endenv keyword}
As the name implies, both keywords \begenv and \endenv are analogous of
\verb|\begin{...}| and \verb|\end{...}| pair in \LaTeX, respectively.
Thus below code
useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@] {
%#|+color|purple@begenv@ center
%#    asdsad
%#|+color|purple@endenv@ center
```

```
}
is exactly same as
useenv Verbatim [numbers=left, numbersep=5pt, frame=single, commandchars=+|@] {
%#|+color|purple@useenv@ center {
%#    asdsad
%#}
}
Then why we need \begenv and \endenv if we already have \useenv*?

\section{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 {\tt\#label}, which will be explained later. Since Vesti
is a typewriting-oriented language, the word \lq\lq label\rq\rq\ is often used in its
ordinary sense rather than in its special semantic meaning within the language.

Followings are reserved as builtin functions.

useenv table [ht] {
    \centering
    #lu:
    local content = read_all("../src/lexer/Token.zig")

    -- match .{ "here" }
    local pat = "%.%{%s*\"([^\"]+)\"%s*%}"

    local builtins = {}
    for name, tok in content:gmatch(pat) do
        builtins[#builtins + 1] = name
    end
    table.sort(builtins)

    vesti.print([[\begin{tabular}{ccccc}]])

    for i, kw in ipairs(builtins) do
        local cell = string.format("\\#\\verb@%s@", kw)
        if (i % 5) == 0 then
            vesti.print(cell .. [[\\]])
        else
            vesti.print(cell .. "&")
        end
    end

    vesti.print([[\end{tabular}]])
    :lu#[readAll]
    \caption{Builtins in Vesti}
}

\section{Source Code of This Document}
Below code was generated by inline lua.
useenv Verbatim [numbers=left, numbersep=5pt, frame=single] {
#lu:
    local content = read_all("vesti_man.ves")
    for line in content:gmatch("([^\r\n]*)\r?\n?") do
```

```
        vesti.print(line)
    end
:lu#[readAll]
}

```