VerTeX User's Manual

Steve Kieffer*

Updated: 18 January 2013

Contents

1 Introduction

`VerTeX` is a pre-filter designed to make typing TeX / LaTeX files easier. For the remainder of this manual, "TeX" will mean TeX and/or LaTeX.

The name `VerTeX` stands for "Verbal TeX", since the language allows you to write TeX in a way that is closer to what you say when you read mathematics aloud (if you ever do that), and because it offers many verbal alternatives to symbolic notations, allowing you to keep your fingers over the home row on the keyboard. The word `VerTeX` is pronounced "ver-tech".

`VerTeX` is a Python script, and if you have the latest (or not even the latest) Python 2.x it should run just fine. I have not tested it on any Python 3.x.

The features of `VerTeX` are:

1. Semiautomatic subscripts and superscripts

   Example: `a1, a2, ddd, an` produces $a_1, a_2, \ldots, a_n$.

*`http://skieffer.info`

2. Few to no backslashes in math mode

   Example: `alpha mapsto beta` produces $\alpha \mapsto \beta$.

3. Font prefixes

   Example: `frp in bbZ` produces $\mathfrak{p} \in \mathbb{Z}$.

4. Keyword delimiting patterns instead of braces

   Example: `frac x over y;` produces $\frac{x}{y}$.

5. Macro definitions on the fly

6. Transparency to ordinary TeX

In particular, transparency to ordinary TeX means that, with only a couple of exceptions (to be covered in Sections 2 and 3 below), a `VerTeX` file could potentially contain nothing but ordinary TeX. You are free to use as many or as few of `VerTeX`'s features as you wish.

Installing and Using `VerTeX`:

Download the Python script `vertex` and save it anywhere on your `PATH`.

You write a `VerTeX` file exactly as you would write a `TeX` (or `LaTeX`) file, except that within math mode [1] you are free to use the `VerTeX` syntax.

You should give your `VerTeX` file the extension `.vtx`. After it passes through the `VerTeX` filter, a file with extension `.tex` will be automatically produced.

You may use the `\input` command to build large documents exactly as you would in ordinary `LaTeX`. Just as you would do there, do *not* put the file extension on the end of the name of the file passed to the `\input` command. However, the file itself should have the `.vtx` extension.

At the command line, type

```
vertex MODE FILENAME
```

where `MODE` is one of the switches `-t`, `-l`, or `-p`, and `FILENAME` is the name of the `.vtx` file to be processed. Note that the `MODE` cannot be omitted. You must choose one of the three.

- `MODE -t`: In this mode `vertex` merely translates your `.vtx` file into a `.tex` file.
- `MODE -l` and `-p`: The translation is performed as in the `-t` mode, and then the output `.tex` file is passed on, either to `latex` (in `-l` mode), or to `pdflatex` (in `-p` mode).

We now proceed to discuss the features of VerTeX.

[1] I.e. any math mode: inline, display, equation environment, what have you.

2 Slash the Backslashes

When you are writing mathematics, how often do you want an 'α', and how often do you want to multiply the variables a, l, p, h, and a together, in that order? Why then should `$alpha$` give you a sequence of Roman letters, while the Greek letter requires a backslash?

In `VerTeX`, `$alpha$` yields 'α', and if you really want the product of variables, simply put spaces between the letters, as in `$a l p h a$`.

In general, `VerTeX` keywords are not strings of letters preceded by a backslash, but simply strings of letters uninterrupted by whitespace.

Conceptually, the keywords in `VerTeX` may be divided into the following four kinds, according to the role they play in avoiding backslashes.

Types of VerTeX keywords:

1. bsme
2. built-in
3. bracket word
4. font prefix

2.1 bsmes

The abbreviation 'bsme' is short for "backslash me", and a bsme keyword is one that is exactly the same as a keyword in `TeX`, except without the backslash. It produces the exact same result as the corresponding `TeX` keyword. Table 1 shows all the bsme keywords in `VerTeX`. Each one produces the same effect in `VerTeX` as if you had put a backslash in front of it and typed it in `TeX`. So for example, `$subseteq$` achieves the same thing in `VerTeX` as `$\subseteq$` does in `TeX`.

You will notice that some of the Greek letters (namely, those whose names are two-letters long) are missing from this list. This is intentional, and is discussed in Section 2.2.

2.2 built-ins

The so-called "built-in" keywords do not correspond to any existing `TeX` keywords. They do not take any arguments, but simply translate directly into some string of `TeX`, and their purpose is to in one way or another give an easier way to type certain commonly used `TeX` strings.

In particular, one of the goals of `VerTeX` is to give you the option to keep your fingers over the home row while typing, and for this reason the built-in keywords provide many alphabetical equivalents to `TeX` strings that ordinarily involve non-alphabetical characters. For example, `inv` (for "inverse") produces `^{-1}`, and `squ` (for "squared") produces `^2`.

mapsto	$\mapsto$	leq	$\leq$	geq	$\geq$	neq	$\neq$
times	$\times$	subseteq	$\subseteq$	supseteq	$\supseteq$	subsetneq	$\subsetneq$
supsetneq	$\supsetneq$	subset	$\subset$	supset	$\supset$	not	$/$
cdot	$\cdot$	det	det	equiv	$\equiv$	cong	$\cong$
sim	$\sim$	cup	$\cup$	cap	$\cap$	nmid	$\nmid$
mid	$\mid$	infty	∞	ker	ker	iff	$\iff$
forall	$\forall$	exists	$\exists$	int	$\int$	sin	sin
cos	cos	log	log	exp	exp	quad	
qquad		alpha	α	beta	β	gamma	γ
delta	δ	epsilon	ϵ	zeta	ζ	eta	η
theta	θ	iota	ι	kappa	κ	lambda	λ
rho	ρ	sigma	σ	tau	τ	upsilon	υ
phi	ϕ	chi	χ	psi	ψ	omega	ω
Gamma	Γ	Delta	Δ	Theta	Θ	Lambda	Λ
Sigma	Σ	Phi	Φ	Psi	Ψ	Omega	Ω
ell	ℓ						

Table 1: bsme keywords

A few of the built-ins, such as `pie` and `ksi`, provide alternate spellings [2] for some of the Greek letters (π and ξ, in this case). All Greek letters whose names are two letters long are respelled in order to avoid collisions with `VerTeX`'s automatic subscripting mechanism, as described in Section 3; those with longer names are given abbreviated respellings simply to allow for faster typing. The built-ins and the `TeX` that they translate to are shown in Table 2.

2.3 bracket words

In `TeX` there are many constructions in which a keyword takes arguments surrounded by braces { }. For example,

```
\frac{\pi}{4}
```

yields

$$\frac{\pi}{4}$$

In `VerTeX` the same construction is achieved by

```
frac pie over 4;
```

[2] If math departments around the world can get away with eating massive amounts of pie every year on March 14th, I think I can get away with this.

ccc	\cdots	ddd	\ldots
vvv	\vdots	equ	=
div	|	plus	+
minus	-	to	\rightarrow
gets	\leftarrow	in	\in
ltn	<	gtn	>
del	\partial	mod	\mathrm{mod}
inv	^{-1}	squ	^2
cubed	^3	star	^*
mult	^\times	empty	\varnothing
plmi	\pm	mipl	\mp
pie	\pi	Pie	\Pi
ksi	\xi	Ksi	\Xi
new	\nu	mew	\mu
eps	\varepsilon	vphi	\varphi
vtheta	\vartheta	vpi	\varpi
sig	\sigma	Sig	\Sigma
lam	\lambda	Lam	\Lambda
gam	\gamma	Gam	\Gamma
alp	\alpha		

Table 2: built-in keywords

In this example, `frac`, `over`, and the final `;` serve as *bracket words*. The idea of course is that if a certain construction takes arguments, then the arguments are to be surrounded by the appropriate bracket words. For the most part, the final bracket word will be a semicolon.

The list of all such constructions in `VerTeX` is given in Table 3. Here, A', B', and C' stand for the `TeX` to which the `VerTeX` A, B, and C would be translated.

2.4 font prefixes

Technically font prefixes are not "keywords" in and of themselves. They are two- or three-character prefixes which, when followed by a letter of the alphabet, produce that letter in the appropriate font. The prefixes and the fonts that they correspond to are listed in Table 4.

Thus, for example, instead of

```
\mathfrak{p} \quad \mathsf{M} \quad \mathbf{v} \quad
\mathbb{Q} \quad \mathcal{O} \quad \mathscr{B}
```

you may type

VerTeX:	is translated into the TeX:	mnemonic / idea
A `of` B `;`	A' `(` B' `)`	function evaluation "f of x
`qnt` A `;`	`\left(` A' `\right)`	"quantity"
`bqnt` A `;`	`\left[` A' `\right]`	"bracket quantity"
`set` A `;`	`\{` A' `\}`	"set A"
`bset` A `;`	`\left\lbrace A' \right\rbrace`	"big set A"
`abs` A `;`	`\left|` A' `\right|`	"absolute value"
`seq` A `;`	`\left\langle` A' `\right\rangle`	"sequence"
`sup` A `;`	`^{` A' `}`	"superscript"
`supp` A `;`	`^{(` A' `)}`	"superscipt in parentheses
`sub` A `;`	`_{` A' `}`	"subscript"
`eth` A `;`	`^{\mathrm{`A'`}}`	ordinals, 1^{st}, 2^{nd}, 3^{rd}, 4^{th},
`pmod` A `;`	`\left(\mathrm{mod}\,` A' `\right)`	"parenthesis mod"
`sqrt` A `;`	`\sqrt{` A' `}`	square root
`bar` A `;`	`\bar{` A' `}`	over bar
`tilde` A `;`	`\tilde{` A' `}`	tilde overscript
`hat` A `;`	`\hat{` A' `}`	hat overscript
`words` A `;`	`\:\mbox{` A' `}\:`	plain text in math mode
`frac` A `over` B `;`	`\frac{` A' `}{` B' `}`	quotient
`root` A `base` B `;`	`\sqrt[`A'`]{`B'`}`	general root
`legen` A `over` B `;`	`\left(\frac{`A'`}{`B'`}\right)`	Legendre symbol
`binom` A `choose` B `;`	`\binom{`A'`}{`B'`}`	binomial coefficient
`map` A `from` B `to` C `;`	A' `:` B' `\rightarrow` C'	map
`sum over` A `;`	`\sum_{`A'`}`	sum
`sum over` A `from` B `to` C `;`	`\sum_{`$A' = B'$`}^{`C'`}`	sum
`prod over` A `;`	`\prod_{`A'`}`	product
`prod over` A `from` B `to` C `;`	`\prod_{`$A' = B'$`}^{`C'`}`	product
`union over` A `;`	`\bigcup_{`A'`}`	union
`union over` A `from` B `to` C `;`	`\bigcup_{`$A' = B'$`}^{`C'`}`	union
`inters over` A `;`	`\bigcap_{`A'`}`	intersection
`inters over` A `from` B `to` C `;`	`\bigcap_{`$A' = B'$`}^{`C'`}`	intersection
`limit over` A `;`	`\lim_{`A'`}`	limit

Table 3: Bracket word constructions

prefix	font
fr	mathfrak
sf	mathsf
bf	mathbf
bb	mathbb
cal	mathcal
scr	mathscr

Table 4: Font prefixes

```
frp quad sfM quad bfv quad bbQ quad calO quad scrB
```

to get

$$\mathfrak{p} \quad \mathsf{M} \quad \mathbf{v} \quad \mathbb{Q} \quad \mathcal{O} \quad \mathscr{B}.$$

3 Semiautomatic subscripts and superscripts

In mathematics, subscripted variables are the coin of the realm, and therefore it ought to be easy to type them. VerTeX makes it fast and easy to get subscripts and superscripts. Table 5 shows a few examples.

TeX	VerTeX	output
a_1, a_2, \ldots, a_{n+1}	a1, a2, ddd, an+1	$a_1, a_2, \ldots, a_{n+1}$
x_{i_1}, x_{i_2}, \ldots, x_{i_m}	xivv1, xivv2, ddd, xivvm	$x_{i_1}, x_{i_2}, \ldots, x_{i_m}$
a_{i j}^2	aijuu2	a_{ij}^2

Table 5: Semiautomatic subscript and superscript examples

The semiautomatic subscripting and superscripting (henceforth SSS) mechanism of VerTeX is very handy, and, as the examples in Table 5 show, makes it much easier to type certain common kinds of subscripts and superscripts.

While many subscript and superscript combinations can be achieved through SSS, some things are not possible. In such cases, one can use the sub and sup keywords as in Table 3, or can even fall back on standard TeX syntax.

The complete description of the SSS process is a bit complex, but for most common purposes it is quite simple. Therefore before we give a detailed specification of the process, we consider the main ideas.

First we need some terminology. In a figure such as B_s^t we of course refer to s and t as subscript and superscript. Let us refer to B as the "base."

In most cases, SSS is simple. VerTeX will take a word ω and split it as $\omega = \beta\sigma$, where β is the longest initial segment of ω that matches as a letter name, and σ is everything that remains. Then β will be made the base, and σ will be the subscript.

Examples:

`pi`	$\rightarrow$	p_i
`alphan`	$\rightarrow$	α_n
`bbZm`	$\rightarrow$	$\mathbb{Z}_m$
`cn+1`	$\rightarrow$	c_{n+1}
`ai,j`	$\rightarrow$	$a_{i,j}$
`zeta`	$\rightarrow$	ζ

There are several things to note about these examples:

1. It was so that `pi` could be available for automatic subscripting that we gave the Greek letter π the (admittedly somewhat silly) spelling `pie`. Writing p_i is a daily occurrence for anyone who works with prime numbers, and this includes a lot of mathematicians.

2. Letters with extended names, like `alpha`, and letters with a font prefix in front of them, like `bbZ`, will indeed be counted as initial letters.

3. Commas, as well as plus and minus signs, are considered part of the word. Read further details about this below.

   NB: this is one way in which `VerTeX` is *not transparent* to ordinary TeX.

4. What happened with `zeta`? Perhaps we were hoping for z_η, but of course `VerTeX` instead matched the entire word `zeta` as the base. There is a way to get around this, which we discuss below. Preview: You may type `zvveta` in order to get z_η.

The details.

To the `VerTeX` parser, a "word" consists of alphanumeric characters, as well as commas and the plus and minus symbols. It must begin with an alphabetical character [3] For those familiar with regular expressions, this means that words are built on the character class

```
[A-Za-z0-9+-,]
```

The last three symbols are included in the character class because they are common in subscripts. **However,** this means that if you do not want to accidentally trigger a subscript, you need to put whitespace on at least one side of these characters. This is one way in which `VerTeX` is **not transparent** to ordinary TeX.

Now suppose that ω is the next word that `VerTeX` has to process. If ω fails to match as any kind of keyword – bsme, built-in, bracket word, or user-defined keyword (see Section 4) – then ω is submitted to the SSS process.

[3] In other words a "letter," but this is one of the letters in the character class `[A-Za-z]`, and is not to be confused with all things considered letters in `VerTeX`, which includes, for example, Greek letters, and letters with font prefixes.

VerTeX first matches the longest possible letter name at the beginning of ω, as we discussed above. Let the word ω consist of initial letter β followed by remainder σ, that is, $\omega = \beta\sigma$. Then β will be the base, and σ will give one or more subscripts and/or superscripts.

In the simplest case, σ simply represents a subscript. It is possible however to switch between subscripts and superscripts using the special character sequences `vv`, `uu`, and `UU`.

A few examples illustrate all of the ways to use these control sequences:

`auur`	$\rightarrow$	a^r
`aiuur`	$\rightarrow$	a_i^r
`auurvvk`	$\rightarrow$	a^{r_k}
`aivvj`	$\rightarrow$	a_{i_j}
`aivvjuur`	$\rightarrow$	$a_{i_j^r}$
`aivvjUUr`	$\rightarrow$	$a_{i_j}^r$
`zvveta`	$\rightarrow$	z_η

Sequence `vv` opens a deeper subscript. In TeX it is as though you typed `_{`.

Sequence `uu` closes a subscript *and* opens a superscript. In TeX it is as though you typed `}^{`.

Seuqnece `UU` closes *two* subscripts and opens a superscript. In TeX it is as though you typed `}}^{`.

An exception is that if `vv` is used at the very beginning of σ, it merely keeps you at the first subscript level. Thus, `zvveta` provides a way to produce z_η, while `zeta` simply gives ζ.

4 Macros on the fly

At any point in a VerTeX document, you may write a special comment line that begins with `%%vtx%%`, signaling that you are going to give a macro definition.

If k is the keyword that you want to define, and if t is the text that it should expand to, then the syntax is as follows:

`%%vtx%%` k `%%` t `%%;`

If you want the keyword to take arguments, simply name the arguments in the expansion text t using `#1`, `#2`, `#3`, and so on. When used, the arguments to the keyword should be separated by semicolons.

Examples:

After the line

```
%%vtx%% NLK %% N sub L | K; %%;,
```

`NLK` expands to `N sub L | K;`, producing $N_{L|K}$.

After the line

```
%%vtx%% atoq %% a sub #1; sup q sub #1; of #2 ;; %%;,
```

`atoq n ; x ;` expands to `a sub n; sup q sub n; of x;;`, producing $a_n^{q_n(x)}$.

5 Transparency to ordinary TeX

In general, it is hoped that 99% of the time ordinary `TeX` will pass through VerTeX unaltered, so that you can use as many or as few of the VerTeX features as you wish.

The main known lack of transparency is that discussed in Section 3 on the SSS mechanism. Namely, you must put whitespace on at least one side of any comma, plus, or minus symbol in a math environment in order to prevent unwanted automatic subscripting.

The other main issue is that the `VerTeX` parser needs to look through your input document in search of math modes, and each new kind of math environment needs to programmed in. While most of the common environments, such as `equation` and AMS `align`, are supported, some environments, such as the `commdiag` environment in the `xypic` package, are not yet supported.

Apart from these points, the author has not yet discovered any other issues while making regular use of the system for several years. Of course, bug reports are welcome.[4]

As a "safety net," any word whatsoever may be prefixed with a double backslash \\, in order to allow that word to pass through `VerTeX` unaltered. To be precise, if there is any remainder w to the word, this, without the two backslashes, is what will pass through. If just two backslashes alone are typed, they will pass though unaltered (which is useful in TeX table environments). Meanwhile, any word beginning with a single backslash is passed through `VerTeX` completely unaltered, i.e. with the leading backslash still intact.

In summary:

$$\begin{array}{lcl} \backslash\backslash w & \mapsto & w \\ \backslash\backslash & \mapsto & \backslash\backslash \\ \backslash w & \mapsto & \backslash w \end{array}$$

where w is a word at least one character long.

[4] Visit `http://skieffer.info`.