ZapoTEX: User's Manual (EARLY DRAFT)

Vincent HUGOT

October 29, 2011

Note: this is a very early draft of a very experimental program...

Contents

1 Introduction

Z*apo*TEX is essentially a pretty-printer which takes `OCaml`, `Prolog` or `B` specification source code as input and outputs corresponding LATEX. It is highly customisable and supports extensive syntax highlighting, and on-the-fly definition of LATEX aliases for common identifiers, operators etc.

Most of this manual focuses on `OCaml`, because it is the first language which was implemented in **Z***apo*TEX. However the same mechanisms apply to any supported language.

1.1 Use Cases

- Writing articles and/or reports which present algorithms written in `Caml`, or simply code snippets. In this case, the code is embedded in the document in the same way mathematical expressions are, and **Z***apo*TEX acts as a preprocessor for the LATEX document. See section 2.
- Limited literate programming in `OCaml`; in that case, LATEX code appears as special comments in the source. Refer to section 3.

1.2 Example Rendered Code

```
(** Some types *)
val fold← : (α → β → α) → α → β list → α
val fold→ : (α → β → β) → α list → β → β
type α seq = Nil | Cons of α × α seq

(** This is a documentation comment... *)
let rec fold← f accu = function
  | [] → accu (* some regular comment *)
  | a::l → fold← f (f accu a) l

(* nested (* comments (* are (* cool *) but *) should *)
  not be abused *)
let rec fold→ f l accu = match l with
  | [] → accu
  | a::l → f a (fold→ f l accu)
```

```
16
17 let some_aliases 𝔵 𝒳 = List.find (fun ξ → ξ ⩽ 𝔵 && ξ ≠ 𝔵 / 2) 𝒳
```

1.3 Help Page of the ZapoTEX Program

```
** ZapoTeX (zapotex)
** by Vincent HUGOT
** email : vincent.hugot@gmail.com
** web   : vincent-hugot.com

Usage (Caml code -> LaTeX prettifier):

--tex 'file.z.tex' > 'file.tex'
  Takes a LaTeX file containing code formulae, and outputs a pure LaTeX file
  on the standard output, which can then be compiled by LaTeX. The code
  is prettified according to the customisable definitions in zapoml.tex.

--ml 'file.ml' > 'file.tex'
  Takes a Caml source file, and outputs corresponding prettified LaTeX version
  which can then be input into a LaTeX document.

--pl 'file.pl' > 'file.tex'                          Same for Prolog sources.
--b  'file.B'  > 'file.tex'                          Same for B sources.

Usage (miscellaneous):

--euroZone,-e 'file1.tex' ... 'fileN.tex'
  Removes all deprecated $$..$$ math constructs in each file and replaces
  them by \[..\]. A backup of each file is made before this operation.
  If --inline is passed before, also does inline replace: $x$ -> \(x\).

--accents,-a    [BROKEN IMPLEMENTATION | DO NOT USE]

--dump-vocabularies
  Outputs LaTeX code detailing the list of all kind definitions, aliases
  and generaliases predefined in ZapoTeX for each supported language.

-strict
  Non-strict mode: don't exit on ZapoTeX command error.

--help          Displays this help page
```

2 Code as equations

To use **Z***apo*TEX in that way, you only need to keep three things in mind:

- ◇ Your document must input the file `zapoml.tex`, which defines how lexical constructs of Caml code (such as types, keywords, operators etc) should be rendered. This file can be tweaked to the user's liking.
- ◇ Enclose `Caml` code between ## tags, in the same way that display math is between $$ in TEX. If the opening tag is ### instead of ##, the code is centred; the closing tag is always ##.
- ◇ To compile, pre-process your document, say, `doc.z.tex`, using **Z***apo*TEX: `zapotex -tex doc.z.tex > doc.tex`. The generated document `doc.tex` is then a perfectly ordinary LATEX file, which can be compiled as usual.

Example: the following LATEX code:

```
\dots note that the type constructor ##Cons of 'a * 'a seq##
is generally denoted\dots
```

is rendered:
. . . note that the type constructor *Cons* **of** $\alpha \times \alpha$ *seq* is generally denoted. . .

Practical notes:

- ◇ I recommend the following naming convention: A LATEX file containing **Z***apo*TEX Caml code should have the extension `.z.tex`, and the product of a run of **Z***apo*TEX on it should be named the same, without the `.z`. So a typical **Z***apo*TEX run should follow the pattern:

  `zapotex -tex` *X*`.z.tex > `*X*`.tex`
- ◇ Note that **Z***apo*TEX is perfectly compatible with both LATEX and PDFLATEX.
- ◇ **Z***apo*TEX's running time is completely negligible before that of LATEX.
- ◇ Note that you *cannot* have your LATEX document d input other documents d_k, $k \in 1..n$, containing **Z***apo*TEX markup. In that case, you need to run **Z***apo*TEX on each of the external documents d_k first, yielding, say, z_k, and make it so that the main document d inputs the z_k instead

of the d_k. This remark is of course irrelevant if you are preparing a short exam subject, as you probably have only a single `.z.tex` file; but in a more complex project, you will need to take that into account. I recommend writing a small shell script to automate the build process. It is also possible to write a LaTeX package using *shell escape* to run **Z***apo*TEX on the fly. I might do it if I ever need that, but that's probably not anytime soon. See Section 7 for related planned features.

3 Literate programming

In that case, I simply mean the ability to write LaTeX into source code comments and have it render correctly. This was supported by DumBeX and PLTeX, but I have not rewritten those features yet. Simply because I have not needed them again. If *you* need them, drop me an email.

4 Kinds, aliases and other beautifications

Keywords of the language and type constructors, for instance, are different objects and should be displayed differently. Furthermore, some identifiers and operators are crude ASCII replacements for Greek letters, mathematical symbols and so on. For instance `'a` should really be α and <> stands for $\neq$. **Z***apo*TEX defines a number of standard aliases (such as these), and enables the user to define her own easily.

4.1 Kinds

Each lexical construct of the language is associated to a *kind*; for instance, it can be a keyword, a variable, a type constructor, etc. Depending on its kind, a lexical element can be typeset differently by **Z***apo*TEX. For instance, by default Caml keywords such as **`open`** are typeset in black, bold face, whereas a type such as *float* is written in a blue, italic sans-serif font. Most of Caml's keywords and types are predefined, but the user can of course add her own using **Z***apo*TEX commands. Consider

```
##let foo x = ... return x##\\
%## keyword return ##%
##let foo x = ... return x##
```

```
let foo x = ... return x
let foo x = ... return x
```
or

```
%## type valuation ; lident valuation ##%
##let eval (valuation : valuation) etc = ...##
```

```
let eval (valuation : valuation) etc = ...
```

Note that here we defined "valuation" to be both a declared type and a lowercase identifier, and **Z***apo*TEX used a heuristic to decide which instance was which, even though such a question cannot be decided at the lexical level in general (**Z***apo*TEX acts as a lexer, not as a parser).

This (a single identifier having two kinds) may seem a very special case, but in fact it arises frequently even within just-out-of-the-box OCaml. For instance, consider the declaration

$$\texttt{let (a : } \mathit{int\ ref}\texttt{) = ref 0}$$

We see here that "`ref`" appears once as a type, an another time as the function $\texttt{ref} : \alpha \to \alpha\ \mathit{ref}$ . . . except that it is typeset as a keyword instead of a lowercase identifier. I chose to consider it a keyword because of its special status as sole constructor of the built-in reference type $\alpha\ \mathit{ref}$. Thus **Z***apo*TEX predefines *ref* as having the two kinds `type` and `keyword`. If you don't like this, do

```
%## rm kind ref; type ref; lident ref ##%
```

and then in the expression [`let (a : int ref) = ref 0`], `ref` appears as a simple identifier instead of a keyword, when it is not a type. Or you could only define it as a type [`let (a : int ref) = ref 0`], or only as a lowercase identifier [`let (a : int ref) = ref 0`], which is equivalent to giving no kind definition for it.

Of course this digression on `ref` merely serves to illustrate kinds; the default settings for `ref` (type or keyword) are quite good in my opinion, and I recommend you leave them alone.

4.2 Basic Beautification

Patterns for primes and subscripts in lowercase identifiers are detected and rendered adequately by **Z***apo*TEX:

```
##x y x_y x_yz x_1 x_23 long_x long_0 long_42 other_text ##\\
##x x' x'' x''' x'''' o'harra o'harra' o'harra'' o'harra'''##\\
##_X _X' _X'' _X''' alpha alpha' alpha''##
```

x y x_y x_yz x_1 x_{23} $long_x$ $long_0$ $long_{42}$ other_text
x x' x'' x''' x'''' o'harra $o'harra'$ $o'harra''$ $o'harra'''$
$\mathcal{X}$ $\mathcal{X}'$ $\mathcal{X}''$ $\mathcal{X}''''$ α α' α''

Note that this integrates with aliases (see below), simple or generalised: for instance there is an alias for `alpha`, but not for `alpha'`. So **Z***apo*TEX tries to find an alias for `alpha'`, finds none, isolates the primes, and tries to find an alias for the prime-less `alpha`, succeeds, and thus renders α'. Thus there is no need to define primed versions of your aliases: the base symbol suffices.

4.3 Simple Aliases

With simple aliases, a given identifier is automatically replaced by some specific LATEX code. Aliases are defined using the **Z***apo*TEX command `alias`, in special comments:

```
%##%
 alias x "new\_x", y "new\_y" ;
 alias math a "a", b "b";
 alias math square "x^2";
 alias bold math cube "x^3"
%##%
##x y a b square cube##
```

is rendered: new_x new_y a b x^2 $\mathbf{x}^3$. Note that aliases can be removed if they become inappropriate in another part of the document:

```
%## rm alias square ##%
```

The same code is now rendered: new_x new_y a b square $\mathbf{x}^3$.

There are about ninety predefined aliases in **Z***apo*TEX; they are all listed in section 6[p10].

4.4 Generalised Aliases

Generalised aliases enable the user to define whole families of aliases in one command. This is done using regular expressions. Let us have fun and say that x"n should be translated into x^n, for any letter x and any number n.

```
%###% galias math POWERS
    "\([A-Za-z]\)''\([0-9]+\)"          "{\1}^{\2}"
%###%
##something other''x other''3 x''3 y''5 A''67 B''C##
```

is rendered as: `something other"x other"3` x^3 y^5 A^{67} `B"C`. In a nutshell, anything that matches the pattern is translated, while the rest is simply dealt with as usual. Just as with simple aliases, generalised aliases *can* be removed. The only difference is that instead of using simply the "left part" of the binding – which was practical for simple aliases as it was a single identifier, but is *not* practical here as we have a nasty regular expression – we will refer to a generalised alias binding by its given identifier, which, in this case, is "POWERS".

```
##X''8##
%## rm galias POWERS ##%
##X''8##
```

Yields X^8 `X"8`. In general, I recommend the convention of always naming generalised aliases in FULL CAPS.

Let us now look at another fun possibility: list enumerations. The aim here is to define a shorthand naming convention for lists whose elements are named and enumerated. Consider the following code and its ZapoTEX rendering:

```
%###% galias ENUM
  "\([A-Za-z]+\)''\([A-Za-z0-9]+\)'\([A-Za-z0-9]+\)"
  "[\1${}_{\text{\2}};\dots;\text{\1}{}_{\text{\3}}$]"
%###%
Let us see: ##q''1'n and p''1'n and something''foo'bar##
```

Let us see: $[q_1;\ldots;q_n]$ **and** $[p_1;\ldots;p_n]$ **and** $[\text{something}_{\text{foo}};\ldots;\text{something}_{\text{bar}}]$.

In practice, we have used this to define a few simple default generalised aliases which provide easy access to special math fonts. For instance:

```
## x _x _X __x __X _'x _'X ##
```

x _x $\mathcal{X}$ __x $\mathbb{X}$ $\mathfrak{x}$ $\mathfrak{X}$

Those default generalised aliases are listed in Section 6.

5 Supported Languages

At the time of writing, **ZapoTEX** supports the following languages:

	OCaml	Prolog	B
code	*ml	*pl	*b
inline	##..##	#pl#..##	#b#..##
display	###..##	#PL#..##	#B#..##
command	%##..##%	%#pl#..##%	%#b#..##%
- comms.	(*#..*)	/*#..*/	/*#..*/

where *code* corresponds to the vocabulary code used in "dump" commands, (see Section 6), *inline* is the **ZapoTEX** markup for inline mode (where `..` represents the actual source code), *display* is the markup for display (centred) mode, and *command* is the markup for **ZapoTEX** commands. Note that kind definitions, aliases, generalised aliases etc are specific to a given language. For instance the aliases of `OCaml` don't mix with those of `Prolog`. They have no reason to. Collectively, this is called the *vocabulary* of a language. Commands which affect a vocabulary (for instance by creating a new alias) will only deal with the vocabulary corresponding to the command environment they are invoked in. For instance `%##keyword foo##%` will only add the keyword "foo" for the `OCaml` language, and `%#pl#keyword foo##%` will do the same thing exclusively for `Prolog`.

Instances of '%' are ignored while in command mode (treated as whitespace), so you can keep all the commands commented (from LaTeX's point of view). This is sometimes convenient if you work in an editor which does syntactic coloration for TeX code. The same applies when command mode is accessed through command comments. Those are comments of a special form (see table) embedded within source code, which are not rendered as LaTeX by **ZapoTEX** but instead are interpreted as **ZapoTEX** commands. Not that this applies regardless of whether the comment appears in a stand-alone source file or within code embedded in a **ZapoTEX** LaTeX file. For instance

```
#B# /*# alias math test "\ds\int_a^b f(x)\;\textrm{d}x"
*/  A >< B /: test ##
```

Renders the following (utterly nonsensical) `B` code:

$$A \otimes B \notin \int_a^b f(x)\,\mathrm{d}x$$

This is functionally equivalent to

```
%## alias math test "\ds\int_a^b f(x)\;\textrm{d}x" ##%
#B# A >< B /: test ##
```

5.1 Sample Prolog Code

```
#pl#
/* a /* b /* c */ d */ e */ % blah some comment
ord_intersect__(>=, H1, T1,  _H2, T2) :-
  ord_intersect_(T2, H1, T1). ##
```

```
/* a /* b /* c */ d */ e */ % blah some comment
ord_intersect__(>=, H1, T1,  _H2, T2) ⊢
  ord_intersect_(T2, H1, T1).
```

5.2 Sample B Code

```
#b#
transitive_reflexive_closure = closure(relation) \/ closure(Relation) &
transitive_closure = closure1(relation) \/ closure1(Relation)&
x : direct >< product <=> x /: parallel || product &
lambda_expression = % x . (x : 1..n /\ K..L | x** 2 - Y**x) & ##
```

$$\text{transitive\_reflexive\_closure} = \text{relation}^* \cup \mathit{Relation}^* \quad \wedge$$
$$\text{transitive\_closure} = \text{relation}^+ \cup \mathit{Relation}^+ \quad \wedge$$
$$x \in \text{direct} \otimes \text{product} \Leftrightarrow x \notin \text{parallel} \parallel \text{product} \quad \wedge$$
$$\text{lambda\_expression} = \lambda\, x \,.\, (x \in [\![1 \,,\, n]\!] \cap [\![K \,,\, L]\!] \mid x^2 - Y^x) \quad \wedge$$

6 Default Vocabularies

Here is the complete list of default (predefined) vocabularies – that is to say kind definitions, aliases and generalised aliases, – as generated by the `--dump-vocabularies` **ZapoTEX** command switch. Of course the user is free to remove any and all aliases she does not like; please refer to sections 4.3 and 4.4 for more information.

Note that similar dumps can be effected at any place within a document by using the `dump` command. For instance `%## dump *b, *pl ##%` will dump the current vocabularies for `B` and `Prolog`.

Language: **B**
Kinds assignment: 31 definitions.
keyword :: (**VARIANT VARIABLES OR VAR ELSE BEGIN INVARIANT THEN BE skip SELECT WHEN MODEL WHERE PRE CHOICE SETS IN MACHINE END IF ELSIF INITIALISATION DEFINITIONS CONSTRAINTS ANY LET OPERATIONS PROPERTIES EITHER CONSTANTS**)
Simple Aliases: 117 bindings.
! → ∀ ; # → ∃ ; % → λ ; & → ∧ ; * → × ; +-> → ⇸ ; +-» → ⤔ ; -> → → ; -» → ↠ ; -> → → ; /: → ∉ ; /<: → ⊈ ; /«: → ⊄ ; /= → ≠ ; /\ → ∩ ; /|\ → ↑ ; : → ∈ ; :: → :∈ ; < → < ; <+ → ◁ ; <- → ← ; <- → ⟵ ; <-> → ↔ ; <: → ⊆ ; «: → ⊂ ; «| → ⩤ ; <= → ≤ ; <=> → ⇔ ; <| → ◁ ; == → ≙ ; ==> → ⟹ ; => → ⇒ ; > → > ; >+> → ⤔ ; >+» → ⤕ ; >-> → ↣ ; >-» → ⤖ ; >< → ⊗ ; >= → ≥ ; BOOL → 𝔹 ; FIN → 𝔽 ; FIN1 → $\mathbb{F}_1$; INT → $\mathbb{Z}_{rep}$; INTEGER → ℤ ; INTER → ⋂ ; NAT → $\mathbb{N}_{rep}$; NAT1 → $\mathbb{N}^+_{rep}$; NATURAL → ℕ ; NATURAL1 → $\mathbb{N}^+$; PI → ∏ ; POW → ℙ ; POW1 → $\mathbb{P}_1$; SIGMA → ∑ ; STRING → 𝕊 ; UNION → ⋃ ; \/ → ∪ ; \|/ → ↓ ; ^ → ⌢ ; aleph → ℵ ; alpha → α ; beta → β ; beth → ℶ ; chi → χ ; daleth → ℸ ; delta → δ ; ell → ℓ ; epsilon → ϵ ; eta → η ; eth → ð ; gDelta → Δ ; gGamma → Γ ; gLambda → Λ ; gOmega → Ω ; gPhi → Φ ; gPi → Π ; gPsi → Ψ ; gSigma → Σ ; gTheta → Θ ; gUpsilon → Υ ; gXi → Ξ ; gamma → γ ; gimel → ℷ ; inter → ⋂ ; iota → ι ; kappa → κ ; lambda → λ ; mho → ℧ ; mu → μ ; nabla → ∇ ; not → ¬ ; nu → ν ; omega → ω ; or → ∨ ; partial → ∂ ; phi → ϕ ; pi → π ; psi → ψ ; rho → ρ ; sigma → σ ; tau → τ ; theta → θ ; union → ⋃ ; upsilon → υ ; varepsilon → ε ; varphi → φ ; varpi → ϖ ; varrho → ϱ ; varsigma → ς ; vartheta → ϑ ; xi → ξ ; zeta → ζ ; {} → ∅ ; | → | ; |-> → ↦ ; |> → ▷ ; |» → ⤐ ; || → ‖ ;
Generalised Aliases: no binding.

Language: **OCaml**
Kinds assignment: 79 definitions.
flow :: (***failwith exit raise invalid_arg***)
type :: (*array list ref open_flag format int64 float char out_channel unit fpclass in_channel int option int32 bool format4 exn string*)
keyword :: (**functor when while initializer mutable struct land downto ref match rec try done object as mod fun type val new false function true asr external to lxor module exception inherit begin in if constraint**

include lsr lsl class virtual end assert for with else lazy private let or then sig do of lor open method and)
Simple Aliases: 95 bindings.
!= → $\neq_{\phi}$; 'a → α ; 'b → β ; 'c → γ ; 'd → δ ; 'e → ϵ ; 'f → ζ ; 'g → η ; 'h → θ ; 'i → κ ; 'j → λ ; 'k → μ ; 'l → ν ; 'm → ξ ; 'n → π ; 'o → ρ ; 'p → σ ; 'q → τ ; 'r → φ ; 's → χ ; 't → ψ ; 'u → ω ; 'v → Γ ; 'w → Δ ; 'x → Θ ; 'y → Λ ; 'z → Ξ ; * → × ; *. → $\times_{.}$; + → + ; +. → $+_{.}$; - → − ; -. → $-_{.}$; -> → → ; / → / ; /. → $/_{.}$; < → < ; <- → ← ; <= → ⩽ ; <> → ≠ ; = → = ; == → $=_{\phi}$; > → > ; >= → ⩾ ; _delta → $\mathbf{\Delta}$; _gamma → $\mathbf{\Gamma}$; _lambda → $\mathbf{\Lambda}$; _omega → $\mathbf{\Omega}$; _phi → $\mathbf{\Phi}$; _pi → $\mathbf{\Pi}$; _psi → $\mathbf{\Psi}$; _sigma → $\mathbf{\Sigma}$; _theta → $\mathbf{\Theta}$; _upsilon → $\mathbf{\Upsilon}$; _xi → $\mathbf{\Xi}$; aleph → $\aleph$; alpha → $\boldsymbol{\alpha}$; beta → $\boldsymbol{\beta}$; beth → $\beth$; chi → $\boldsymbol{\chi}$; daleth → $\daleth$; delta → $\boldsymbol{\delta}$; ell → ℓ ; epsilon → $\boldsymbol{\epsilon}$; eta → $\boldsymbol{\eta}$; eth → $\boldsymbol{\eth}$; fold_left → $\text{fold}^{\leftarrow}$; fold_right → $\text{fold}^{\rightarrow}$; gamma → $\boldsymbol{\gamma}$; gimel → $\gimel$; iota → $\boldsymbol{\iota}$; kappa → $\boldsymbol{\kappa}$; lambda → $\boldsymbol{\lambda}$; mho → $\mho$; mu → $\boldsymbol{\mu}$; nabla → $\boldsymbol{\nabla}$; nu → $\boldsymbol{\nu}$; omega → $\boldsymbol{\omega}$; partial → $\boldsymbol{\partial}$; phi → $\boldsymbol{\phi}$; pi → $\boldsymbol{\pi}$; psi → $\boldsymbol{\psi}$; rho → $\boldsymbol{\rho}$; sigma → $\boldsymbol{\sigma}$; tau → $\boldsymbol{\tau}$; theta → $\boldsymbol{\theta}$; upsilon → $\boldsymbol{\upsilon}$; varepsilon → $\boldsymbol{\varepsilon}$; varphi → $\boldsymbol{\varphi}$; varpi → $\boldsymbol{\varpi}$; varrho → $\boldsymbol{\varrho}$; varsigma → $\boldsymbol{\varsigma}$; vartheta → $\boldsymbol{\vartheta}$; xi → $\boldsymbol{\xi}$; zeta → $\boldsymbol{\zeta}$;
Generalised Aliases: three bindings.
math : MATHCAL :: "_\([A-Z]\)$" → "\mathcal{\1}" ;
math : MATHBB :: "__\([A-Z]\)$" → "\mathbb{\1}" ;
bmath : MATHFRAK :: "_'\([A-Za-z]\)$" → "\mathfrak{\1}" ;

Language: **Prolog**
Kinds assignment: 137 definitions.
flow :: (***nl term_str msgsend free errorlevel project nobreak db_btrees format trap heap findall save msgrecv fail chain_insertz not write readterm error code printermenu chain_inserta assertz retractall retract term_replace check_determ asserta writef assert chain_insertafter term_bin config bgifont bound chain_terms diagnostics db_chains ref_term gstacksize consult nowarnings bgidriver***)
type :: (*ulong ushort ref byte word char real integer sbyte dword symbol string long unsigned short binary*)
keyword :: (**SINGLE implement STATIC multi OR endclass struct DETERM REFERENCE database elsedef FACTS protected NOCOPY erroneous object as procedure failure ifndef language global abstract IMPLEMENT false enddef ALIGN DOMAINS facts true AS single CLASS CLAUSES static PROTECTED is ERRONEOUS determ goal this if PROCEDURE ENDCLASS IFDEF PREDICATES include DATABASE**

```
ELSEDEF nocopy align class predicates nondeterm FAILURE STRUCT constants
IF LANGUAGE ABSTRACT ifdef OBJECT MULTI reference IFNDEF or GLOBAL GOAL
THIS domains AND ENDDEF and NONDETERM clauses CONSTANTS INCLUDE )
Simple Aliases:  70 bindings.
```

* → × ; + → + ; - → − ; -> → ⟶ ; -> → → ; / → / ; :- → ⊢ ; < → < ; = → = ; =:= → $=_\beta$; =< → ⩽ ; == → $=_\eta$; =\= → $\neq_\beta$; > → > ; >= → ⩾ ; @< → $<_\alpha$; @=< → $\leqslant_\alpha$; @> → $>_\alpha$; @>= → $\geqslant_\alpha$; \= → ≠ ; \== → $=_\eta$; aleph → ℵ ; alpha → α ; beta → β ; beth → ℶ ; chi → χ ; daleth → ℸ ; delta → δ ; ell → ℓ ; epsilon → ϵ ; eta → η ; eth → $\eth$; gDelta → Δ ; gGamma → Γ ; gLambda → Λ ; gOmega → Ω ; gPhi → Φ ; gPi → Π ; gPsi → Ψ ; gSigma → Σ ; gTheta → Θ ; gUpsilon → Υ ; gXi → Ξ ; gamma → γ ; gimel → ℷ ; iota → ι ; kappa → κ ; lambda → λ ; mho → ℧ ; mu → μ ; nabla → ∇ ; nu → ν ; omega → ω ; partial → ∂ ; phi → ϕ ; pi → π ; psi → ψ ; rho → ρ ; sigma → σ ; tau → τ ; theta → θ ; upsilon → υ ; varepsilon → ε ; varphi → φ ; varpi → ϖ ; varrho → ϱ ; varsigma → ς ; vartheta → ϑ ; xi → ξ ; zeta → ζ ;

```
Generalised Aliases:  three bindings.
math : MATHCAL :: "cc_\([A-Z]\)$" → "\mathcal{\1}" ;
math : MATHBB :: "bb_\([A-Z]\)$" → "\mathbb{\1}" ;
bmath : MATHFRAK :: "ff_\([A-Za-z]\)$" → "\mathfrak{\1}" ;
```

7 Planned Features

- ◇ change UI to style "–cmd in out" instead of "–cmd in > out"
- ◇ building on that, add "–make" that compiles all **Z***apo*TEX files and all source files in the directory (option: recursively). Aim: make it easy to handle large projects.
- ◇ Even better: using –shell-escape to do that directly on the fly from within LaTeX. That is to say, write LATEX package so that markup such as ## executes **Z***apo*TEX appropriately.
- ◇ fake LATEX version of markup: to compile **Z***apo*TEX files with LATEX even without **Z***apo*TEX on the system. Translators fake ↔ real markups.
- ◇ Even better, write LATEX package so that markup such as ## starts verbatim modes. This can be a fall-back to the ideal case (on-the-fly

Z*apo*TEX) if –shell-escape cannot be enabled.

- ◇ strict optional mode for evaluation of commands.
- ◇ literate programming.
- ◇ Make generalised aliases more powerful by adding recursive calls. That way indices and primes can be handled in all generality for all languages by a couple of generaliases instead of an ugly kludge in `decorate`. Avoids *n*-plication of the kludge for *n* languages, and lets the user remove/alter this behaviour on the fly.