Standard Z-LaTeX style explained
Community Z Tools (CZT)

Leo Freitas
leo@cs.york.ac.uk

Department of Computer Science
University of York, YO10 5DD
York, United Kingdom

September 2008

Contents

List of Tables

1 Introduction

In this document, $\not\in$ we present a guide to the *Community Z Tools* (CZT) [?] style file (`czt.sty`). It is used to typeset ISO Standard Z notation [?] that is machine readable by CZT tools.

The guide present all the Standard Z characters, as provided in the *Community Z Tools* (CZT) [?] `zchar.xml` file (from the `corejava` project within the SVN distribution). It implements the Unicode rendering and lexis as given in [?, Ch. 6–7]. In what follows, each section corresponds to the XML groups within this XML file. Before we start, let us introduce some context and design decisions within the CZT Standard Z style file (`czt.sty`).

The structure presented in this guide follows the structure presented in Standard Z for lexing, markup directives processor, and parsing. More details about all these symbols and their LATEX rendering can be found in [?, Appendix A]. For easy of reference, we mention at each table caption which part of that appendix symbols are related to. We summarise them all in the end of this document. Furthermore, some characters listed come from the mathematical toolkits, as defined in [?, Appendix B]. We add reference to them and the toolkit files within CZT where they come from. Mathematical toolkit files for Standard Z can be found within the CZT distribution under the `parser` project [?] in its `lib` directory.

Also, CZT lexing/parsing strategy is so that all markup formats are translated to a Unicode stream, which is then lexed/parsed according to the Standard Z concrete syntax grammar [?, Ch. 8]. This way, we only need to have one parser and various markup translators, which reduces the work considerably. Unicode is chosen as a target (among other reasons) because it is an international ISO Standard for lexing. Now, that decision implies in some differences in rendering, as one would expect. For instance, subscripting, which in LATEX is done with “`\_`”, is represented in Unicode with so called *word glues*.

Similarly, whitespace and hard space are also treated differently: in LaTeX hard spaces are typeset as "~", whereas in Unicode they are just normal spaces. Thus, as this document is only concerned with LaTeX markup, word glues and Unicode considerations will not be discussed. On the other hand, LaTeX specific issues, such as hard spaces, will be explained.

1.1 Design decisions

The main design decision behind this document follows CZT guideline that "what you type is what you model". That is, the document "as-is" becomes the source Standard Z (LaTeX) specification to be processed by tools. Other design decisions included: i) keep the style file as minimal, simple, and consistent as possible; ii) document and acknowledge macro definition choices and their origin (when different); iii) normalise definitions for consistency; iv) complete missing cases with either normative rules from the Standard or using common sense; v) keep the style file well documented, but not verbose; and vi) follow order of definitions from Z Standard document.

As the `czt.sty` may be used by both language extensions and LaTeX users, we also provided and explained a series of useful macros for LaTeX rendering that bear no relation with the Standard or the tools. They are useful for LaTeX typesetting only, and are explained in Section 1.2, and Section 8.

1.2 `czt.sty` package options and few useful commands

The `czt.sty` has few options, which are described below:

1. `mathit`: Latin letters in italic shape when in math mode;
2. `mathrm`: Latin letters in roman shape when in math mode;
3. `lucida`: use Lucida Bright fonts (*e.g.,* `lucidabr.sty`);
4. `tkkeyword`: make some toolkit names render as keywords;
5. `color`: typeset Z-LaTeX using colours;
6. `colour`: synonym for `color`;
7. `cntglobally`: count Z definitions globally only;
8. `cntbychapter`: count Z definitions per Chapter and globally;
9. `cntbysection`: count Z definitions per Section and globally.

The default option for when the `czt.sty` is loaded is `mathit, cntglobally`. To change it to have `colour`ful `lucida` fonts, you can load it with

```
\usepackage[colour,lucida]{czt}
```

AMS fonts are used when Lucida Bright is not loaded. For more information about Z declaration counting see Section 7 below. This file is typeset using the following package inclusion:

```
\usepackage[colour,cntbysection]{czt}
```

A few style parameters affect the way Z text is set out; they can be changed at any time if your taste doesn't match mine.

Other useful macros might be used in order to change the various space adjustment registers. They are detailed below, and were inherited from Mike Spivey's `zed.sty`.

`\zedindent` The indentation for mathematical text. By default, this is the same as `\leftmargini`, the indentation used for list environments.

`\zedleftsep` The space between the vertical line on the left of schemas, etc., and the maths inside. The default is 1em.

`\zedtab` The unit of indentation used by `\t`. The default is 2em.

`\zedbar` The length of the horizontal bar in the middle of a schema. The default is 6em.

`\zedskip` The vertical space inserted by `\also`. By default, this is the same as that inserted by approximately `0.5\baselineskip`.

Finally, two other macros that might be frequently used are those for marking commands as either Z-words (text, `$\zword{text}$`) or Z-keywords (**text**, `$\zkeyword{text}$`). They are useful in rendering user defined LaTeX commands, usually present in Z-LaTeX markup directives, as shown in many examples below. We also offer another `\ztoolkit` command, which renders toolkit names, such as dom or ran, wither as `\zkeyword` or `\zword` depending on the option passed.

1.3 Background

This document depends on the style file containing all the definitions for Standard Z (`czt.sty`). It is inspired in the work of many others (see Section 9. By design, the resulting `czt.sty` is to be minimal, yet encompassing of the whole normative LaTeX markup from the Z Standard.

Although all other style files available worked well with various Z tools, they included a considerable amount of code that seemed unrelated to the Standard itself. For instance, presumably for backward compatibility, there were many characters for *Fuzz*, Mike Spivey's Z typechecker at Oxford University [**?**]. Another example are formatting for special formulas within `\mathinner` mathematical operator class (see [**?**, 8.9]).

That meant these style files sometimes created conflicts when used with other (newer) LaTeX packages. For instance, because *Fuzz* uses rather old LaTeX 2.09 (*e.g.,* LaTeX symbols font `lasy`), some conflicts arise when using `zed.sty` (Jim

Davies' style file used in [?]) and AMS fonts. We hope that, with time, any particular backward compatibility issue get solved with a separate (extension) of the base `czt.sty` file.

These additions may be useful for some specific Z tools or editors, or indeed for beautification of the LaTeX document itself. Nevertheless, they cannot be parsed by the Standard Z lexis, hence would produce errors if processed by CZT tools. As LaTeX documents are meant to be machine-readable, such extensions seem outside the scope of CZT's aim. Again, if required, they can be incorporated by the specific users of the feature whom does not observe this machine-readability restriction.

1.4 Document structure

We organise this document following the specific parts within the Z Standard it is related to. We divided sections according to the Z lexis and mathematical toolkits, with a few extra sections for varied material.

We tried to present, as exhaustively as possible, the use of every one of such commands with LaTeX markup typeset in verbatim mode for clarity and reference. We summarise them all in Appendix A. More details can be found at the `czt.dvi` file generated with the `docstrip` utility on the `czt.dtx` document from the CZT distribution.

2 Digit

Loaded automatically by LaTeX (0–9) in whichever font selected, hence no extra work is needed here.

3 Letters

The Z Standard enables users to instruct the parser to recognise new LaTeX commands as part of the Z lexis via the use of markup directives [?, A.2.3], They are typeset as special LaTeX comments `%%Zxxxchar` or `%%Zxxxword`, where "`xxx`" can be either: `pre` for prefix names; `pos` for posfix names; `in` for infix names; and empty for nofix names. Their syntax (accepted by the parser) expects two arguments: the first is the LaTeX command it represents, whereas the second determines how this command is to be rendered in Unicode. Thus, in order to add mathematical symbols as markup directives, one needs to know its corresponding Unicode character (number), which can be found in the Unicode chars [?].

From `prelude.tex`, the Standard Z file containing LaTeX markup directives for Z keywords and basic declarations, all markup directives given as `%%Zprechar` or `%%Zposchar` have special spacing as a pre/posfix operator, which in `czt.sty` is typeset with the `\zpreop` and `\zpostop` macros, respectively. Also, all `%%Zinchar` have special spacing as an infix operator, which can be spaced as either a binary relation with the `\zbinop` macro, or as a relational predicate operator with the `\zrelop` macro. Other `%%Zchar` directives (*e.g.,* Δ, Ξ) do not require special spacing—in the Standard hard spacing is treated differently for them (see [?, A.6.28.2]). The `%%Zxxxword` markup directives are treated similarly.

3.1 Latin

Usual letters (`A`–`Z`, `a`–`z`) are loaded automatically by LaTeX in whichever font selected. Moreover, in mathematical mode, Latin letters are rendered with either italics or roman shape. This depends on the package option selected (see Section 1.2), where italic shape is the default.

3.2 Greek

The Greek letters used in Z are given in Table 1. The last two columns show how characters are rendered with the given LaTeX markup on its side. The last row contains a name convention for framing schemas used in Z promotion [?, Ch. 13] and have no semantic meaning. The spacing for λ and μ changed, as they are prefix keywords in Z for function abstraction and definite description, respectively.

Description	**Role**	**Rendering**	**LaTeX**
Capital Delta	schema inclusion	Δ	`\Delta`
Capital Xi	schema inclusion	Ξ	`\Xi`
Small theta	schema bindings	θ	`\theta`
Small lambda	function abstraction	λ	`\lambda`
Small mu	definite description	μ	`\mu`
Capital Phi	schema promotion	Φ	`\Phi`

Table 1: Greek letters used in Z (A.2.4.1)

The `prelude.tex`define a few other letters as markup directives [?, A.2.3], hence can also be used as variable names that are recognised by the parser, as given in Table 2. Similarly, few capital Greek letters are defined and given in Table 3.

3.3 Other letter

The other letters used in Z are given in Table 4. Note LaTeX subscripting markup has no word glues (see Section 4.2). Also, as $\mathbb{P}$ is defined with the `%%Zprechar` markup directive, it is rendered with appropriate spacing as a prefix keyword. The same applies for finite subsets ($\mathbb{F}$) and their non-empty (1-subscripted) versions (*e.g.,* $\mathbb{P}_1$, $\mathbb{F}_1$). In `number_toolkit.tex`(see Section 5.2.5) and `set_toolkit.tex`(see Section 5.2.2) a few other markup directives also require special LaTeX markup as letters, and is given in Table 5. We also add extra ones for rational and real numbers, as well as boolean values. As they are

Description	**Role**	**Rendering**	**LaTeX**
Small alpha	ordinary name	α	`\alpha`
Small beta	ordinary name	β	`\beta`
Small gamma	ordinary name	γ	`\gamma`
Small delta	ordinary name	δ	`\delta`
Small epsilon	ordinary name	ϵ	`\epsilon`
Small zeta	ordinary name	ζ	`\zeta`
Small eta	ordinary name	η	`\eta`
Small iota	ordinary name	ι	`\iota`
Small kappa	ordinary name	κ	`\kappa`
Small nu	ordinary name	ν	`\nu`
Small xi	ordinary name	ξ	`\xi`
Small pi	ordinary name	π	`\pi`
Small rho	ordinary name	ρ	`\rho`
Small sigma	ordinary name	σ	`\sigma`
Small tau	ordinary name	τ	`\tau`
Small upsilon	ordinary name	υ	`\upsilon`
Small phi	ordinary name	ϕ	`\phi`
Small chi	ordinary name	χ	`\chi`
Small psi	ordinary name	ψ	`\psi`
Small omega	ordinary name	ω	`\omega`

Table 2: Small Greek letters (B.2, `prelude.tex`)

not part of any toolkit, they are not recognised by the parser. Nevertheless, to amend that one just needs to add the following markup directives with their corresponding Unicode character hex-numbers.

```
%%Zchar \rat  U+2119
%%Zchar \real U+211A
%%Zchar \bool U-0001D539
```

4 Special

In this section, we present a list of special characters used in Z. As noted in [**?**, A.2.4.3], "no space characters need to be present around special characters, but it may be rendered if desired."

4.1 Stroke characters

Strokes are summarised in Table 6. Note that ′ (`\prime`) is not used in LaTeX and $'$ (`'`) is used in variables representing after state instead, whereas in Unicode ′ is the one to use! That has to do with backward compatibility and issues related to Unicode.

Description	**Role**	**Rendering**	**LaTeX**
Capital Gamma	ordinary name	Γ	`\Gamma`
Capital Theta	ordinary name	Θ	`\Theta`
Capital Lambda	ordinary name	Λ	`\Lambda`
Capital Pi	ordinary name	Π	`\Pi`
Capital Sigma	ordinary name	Σ	`\Sigma`
Capital Upsilon	ordinary name	Υ	`\Upsilon`
Capital Phi	ordinary name	Φ	`\Phi`
Capital Psi	ordinary name	Ψ	`\Psi`
Capital Omega	ordinary name	Ω	`\Omega`

Table 3: Capital Greek letters (B.2, `prelude.tex`)

Description	**Role**	**Rendering**	**LaTeX**
Blackboard bold A	base numbers	$\mathbb{A}$	`\arithmos`
Blackboard bold N	naturals	$\mathbb{N}$	`\nat`
Blackboard bold P	power set	$\mathbb{P}\_$	`\power`
Blackboard bold F	finite power set	$\mathbb{F}\_$	`\finset`

Table 4: Other letters (A.2.4.2, B.3.6, `prelude.tex`, `set_toolkit.tex`)

4.2 Word glues

Differently from Unicode, in LaTeX, sub and superscripting markup has no word glues (see [?, A.2.4.3]). Instead, the usual LaTeX symbols are used, and no special rendering is needed for super (`^`) and subscripting (`_`).

4.3 Brackets

Table 7 shows all the brackets used in Standard Z. The first two, parenthesis and square brackets, follow the usual LaTeX spacing, whereas the last two (binding and free type brackets) should be treated as `\mathopen/close` LaTeX math operators, hence having a hard space around them. In Z mode, the curly bracket should be treated as a `\mathopen/close` as well, since it is part of set constructors. As curly braces are such low-level TeX, I could not find a way to go around this and just suggest the user to add the hard spaces manually (*e.g.,* `\{~` and `~\}`) as needed. This has no semantic difference, and is just for (personal) aesthetic reasons. Strangely, underscore is grouped at this table in the Standard. It serves both as part of a Z name or as a variable argument (`\varg`) in a definition. For variable arguments, both forms (`\_` and `\varg`) are acceptable by CZT tools.

4.4 Box drawing characters

Table 8 lists the box drawing characters used to render various Z paragraphs, such as axiomatic definitions, schemas, and their generic counterparts, as well

Description	**Role**	**Rendering**	**LaTeX**
Blackboard bold Q	rationals	$\mathbb{Q}$	`\rat`
Blackboard bold R	reals	$\mathbb{R}$	`\real`
Blackboard bold B	boolean	$\mathbb{B}$	`\bool`

Table 5: Extra letters that may be used in Z

Description	**Role**	**Rendering**	**LaTeX**
Prime	after var.	$'$	`'`
Shriek	outputs	!	`!`
Query	inputs	?	`?`

Table 6: Special characters (A.2.4.3)

as section headers. These box drawings characters are used for rendering the various Z-LaTeX environments, as given in Table 19 at Section 6.

4.5 Other special characters

The other special characters from the Z Standard are hard space and new line [**?**, A.2.2]. As LaTeX provide rather fine grained spacing control, various LaTeX commands correspond to the `SPACE` Unicode markup, as summarised in Table 9. Also, note the difference between LaTeX whitespace (*i.e.,* those used to separate LaTeX tokens in math mode) and Z-LaTeX white (or hard) spaces (*i.e.,* those used to separate Z tokens). Thus, ASCII characters for space, tab, and new line are "soft", render as nothing and are not converted to any Z character. On the other hand, Z-LaTeX hard space markup renders as specific quantities of space and is converted according to Table 9. The tab stops counter goes up to 9 (*i.e.,* `\t1` ... `\t9`).

From LaTeX, such mathematical spacing is regulated by the commands and skip values defined in Table 10. To illustrate how to use these skip amount counters, we provide the following LaTeX code, which expands the skip amounts and then restores then back to their default value.

Description	**Role**	**Rendering**	**LaTeX**
Left parenthesis	grouping	(	`(`
Right parenthesis	grouping	)	`)`
Left square bracket	various	[	`[`
Right square bracket	various	]	`]`
Left curly bracket	sets	{␣	`\{~`
Right curly bracket	sets	␣}	`~\}`
Left binding bracket	sets	⦇	`\lblot`
Right binding bracket	sets	⦈	`\rblot`
Left double angle bracket	free types	⟪	`\ldata`
Right double angle bracket	free types	⟫	`\rdata`
Underscore	var. names	⇒ _ ⇐	`\_`
Op. template	var. argument	⇒ _ ⇐	`\varg`

Table 7: Bracket characters (A.2.4.3)

Description	**Role**	**Rend.**	**LaTeX**	**Unicode**
Light horizontal	para boxes	—	N/A	U+2500
Light down	para boxes	|	N/A	U+2577
Light down right	para boxes	┌	N/A	U+250C
Double horizontal	genpara boxes	N/A	N/A	U+2550
Vertical line	box rendering	|	`\mid`	U+007C
Paragraph separator	para marker	└	N/A	U+2514
Paragraph separator	para marker	|	`\where`, `|`	U+007C

Table 8: Boxing characters (A.2.6, A.2.7)

```
% Save original spacing on new skip counter
\newmuskip\savemuskip
\savemuskip=\thinmuskip

Formula with default spacing \hfill $ x \, y \, z $

% Change original spacing
\thinmuskip=20mu

Formula with $20$mu skip \hfill     $ x \, y \, z $

% restore default spacing
\thinmuskip=\savemuskip

Formula with default spacing \hfill $ x \, y \, z $
```

Formula with default spacing $x\,y\,z$

Description	**Role**	**Rendering**	**LaTeX**
Inter word space	hard space	$\Rightarrow~\Leftarrow$	`~`
Inter word space	hard space	$\Rightarrow\ \Leftarrow$	`\␣`
Thin space	hard space	$\Rightarrow\,\Leftarrow$	`\,`
Medium space	hard space	$\Rightarrow\:\Leftarrow$	`\:`
Thick space	hard space	$\Rightarrow\;\Leftarrow$	`\;`
Tab stop 1	hard space	$\Rightarrow\quad\Leftarrow$	`\t1`
Tab stop 2…	hard space	$\Rightarrow\qquad\Leftarrow$	`\t2`

Table 9: Hard space characters (A.2.2)

Description	**Skip counter**	**Space command**	**LaTeX**
Thin space skip	`\thinmuskip`	`\thinspace`	`\,`
Medium space skip	`\medmuskip`	`\medspace`	`\:`
Thick space skip	`\thickmuskip`	`\thickspace`	`\;`

Table 10: Fine control of skip amount for space characters

Formula with 20mu skip $\qquad x \quad y \quad z$

Formula with default spacing $\qquad x\,y\,z$

Similarly, we also have various characters for new lines, and formulae and page breaks, as shown in Table 11.

Description	**Role**	**Rendering**	**LaTeX**
Carriage return	new line	(not shown)	`\\`
Small vertical space	new line	(not shown)	`\also`
Med. vertical space	new line	(not shown)	`\Also`
Big vertical space	new line	(not shown)	`\ALSO`
Small formula break	vert. space	(not shown)	`\zbreak`
Med. formula break	vert. space	(not shown)	`\zBreak`
Big formula break	vert. space	(not shown)	`\ZBREAK`
New page	new page	(not shown)	`\znewpage`

Table 11: New line and break characters (A.2.2)

5 Symbols

List of symbol characters are divided in core and toolkit symbols. The former are related to basic characters and keywords, whereas the latter is related to the Z mathematical toolkit [?, Appendix B].

5.1 Core symbols

Many of the core symbols in LaTeX come directly from the currently selected font, whereas others have special commands. We list them all in Table 12, where expected arguments and their rendering position are given with "_" (`\varg`). The ampersand (`\&`) is needed in (the not so used) mutually recursive free

Description	**Role**	**Rendering**	**LaTeX**
Bullet	set/pred separator	$\bullet$, $\bullet$	`@`, `\spot`
Ampersand	recursive free types	_ & _	`\&`
Right tack	conjecture	$\vdash$ _	`\vdash`
Wedge	logical and	_ $\land$ _	`\land`
Vee	logical or	_ $\lor$ _	`\lor`
Right double arrow	logical implication	_ $\Rightarrow$ _	`\implies`
L/R double arrow	logical equivalence	_ $\Leftrightarrow$ _	`\iff`
Not sign	logical negation	$\neg$ _	`\lnot`
Inverted A	universal quant.	$\forall$ _ $\bullet$ _	`\forall`
Reversed E	existential quant.	$\exists$ _ $\bullet$ _	`\exists`
$\exists$ subscript 1	unique existence	$\exists_1$ _ $\bullet$ _	`\exists_1`
Pertinence	set membership	_ $\in$ _	`\in`
Math. `\times`	cartesian product	_ $\times$ _	`\cross`
Inverted solidus	schema hiding	_ $\setminus$ _	`\hide`
Upwards harpoon	schema projection	_ $\upharpoonright$ _	`\project`
Big fat semicolon	schema composition	_ ⨟ _	`\semi`
Double greater than	schema piping	_ $\gg$ _	`\pipe`
Big fat colon	typechecked term	_ ⦂ _	`\typecolon`

Table 12: Core symbols (A.2.4.4)

types. Its syntax is described in [?, 8.2], whereas its semantics is given in [?, 14.2.3.1]. The fat `\spot` also makes `@` active in math mode so that it gets the right `\mathrel` spacing. Wedge and Vee are the AMS terms for the logical operators.

Other core symbols, such as "/", "; ", ":", ",", ".", "+", "=", *etc.*, are typeset and spaced just as in LaTeX. The symbol for schema projection ($\upharpoonright$, `\project`) is reused for sequence filtering in the toolkit defined in `sequence_toolkit.tex`(see Table 18 in Section 5.2.6). Also, the symbol for schema composition (⨟, LaTeX `\semi`, and Unicode character U+$2A1F$) is very similar (but slightly bigger) than the symbol for relational composition (⨾, `\comp`, U+$2A3E$). Type checked markup is usually given with a big fat colon beside it (⦂, `\typecolon`, U+2982). It is a binary operator with the expression in one side and its type on the other.

Note that spacing with LaTeX infix binary mathematical operators are rendered differently in the presence of new lines in between them.

```
\begin{zed}
  A ~~==~~ S \cup T \\        % usual
  B ~~==~~ S \cup \\ \t2 T \\ % spacing after \cup symbol
  C ~~==~~ S \cup{} \\ \t2 T  % correction
\end{zed}
```

$$\begin{array}{lll} A & == & S \cup T \\ B & == & S\cup \\ & & \quad T \\ C & == & S \cup \\ & & \quad T \end{array}$$

So, when breaking lines near such operators, one need to add the usual LaTeX marker for such situations, as illustrated below (see [**?**, p.525, Table 8.7] for more details), new lines may change the spacing behaviour of infix binary mathematical operators, as the example above shows.

5.2 Toolkit symbols

This section introduces all the characters used within `standard_toolkit.tex`, as mentioned in [**?**, Appendix B]. It has been divided in subsections according to the various Z sections defined in the Standard.

5.2.1 `prelude.tex` and Z keywords

The `prelude` section is an implicit parent of every other section. It assists in defining the meaning of number literal expressions [**?**, 12.2.6.9] and the list arguments of operator templates [**?**, 12.2.12] via syntactic transformation rules. In Table 13, we present the list of symbols and Z keywords (and their fixture) defined in the prelude with markup directives. Z sections enable the user to define self contained named modules with (non cyclic) parent relationships given as a (possibly empty) list of section names. Conditional (**if** − **then** −**else**) allows one to test a predicate which yields an expression depending whether the predicate is *true* or *false*. Let definitions (**let**) allow local variable scoping for expressions.

Operator templates [**?**, C.4.13] have syntactic significance only: they tell the reader how to interpret the template associativity, and how it is rendered as prefix, infix, posfix or nofix. There are three categories of operator templates the user can define: `\function`, for application expressions as *e.g.,*

$$S \cup T = (\_ \cup \_)\,(S, T)$$

`\relation`, for relational predicates as *e.g.,*

$$S \subseteq T = (S, T)(\_ \subseteq \_)$$

Description	**Role**	**Rendering**	**LaTeX**
Z section marker	prefix keyword	**section** _	`\SECTION`
Z section parent	infix keyword	**parents** _	`\parents`
Conditional	prefix keyword	**if** _	`\IF`
Conditional	infix keyword	_ **then** _	`\THEN`
Conditional	infix keyword	_ **else** _	`\ELSE`
Let definition	prefix keyword	**let** _ == _ • _	`\LET`
Application expr.	prefix op. template	**function** _ _	`\function`
Relational pred.	prefix op. template	**relation** _ _	`\relation`
Generic expr. inst.	prefix op. template	**generic** _ _	`\generic`
Left associative	infix op. template	**leftassoc**	`\leftassoc`
Right associative	infix op. template	**rightassoc**	`\rightassoc`
Schema precondition	prefix keyword	**pre** _	`\pre`
List of arguments	infix op. template	$\Rightarrow$, , $\Leftarrow$	`\listarg`
Variable argument	infix op. template	$\Rightarrow$ _ $\Leftarrow$	`\varg`
Boolean truth	ordinary name	**true**	`\true`
Boolean falsehood	ordinary name	**false**	`\false`

Table 13: `prelude.tex`symbols (A.2.4, B.2)

and `\generic`, for generic instantiation of expressions as *e.g.,*

$$X \leftrightarrow Y, \qquad \varnothing[\mathbb{N}]$$

Application expressions (`\function`) are used for both fixed (as pre, in, or pos fixed) function operator application (*e.g.,* infix $S \cup T$), and as its equivalent (*e.g.,* nofix $(\_ \cup \_)(S, T)$) version. Relational (or membership) predicates (`\relation`) are used for both set membership (*e.g.,* $x \in S$), equality (*e.g.,* $S = T$), and as an operator that is a predicate (*e.g.,* $S \subseteq T$). Generic instantiation expressions are used for generic operator application as in when building relation ($X \leftrightarrow Y$) or function ($X \rightarrow Y$) spaces.

Furthermore, all infix `\function` and `\generic` operator templates must have two explicit declarations: one for their binding power, which is as a natural number (the higher the number tighter the precedence); and one for their (left or right) associativity. They are used to resolve operator precedences. For instance, $S \cup T \cap R = (S \cup (T \cap R))$ because $\cap$ binds tighter than $\cup$ (see binding powers in Table 14 below). Relational predicates and prefix, posfix and nofix function and generic operators do not have precedence or associativity explicitly given. Examples of this notation can be found in [**?**, Appendix B], and are highlighted in the **Role** column in the tables below for each operator template defined in the standard toolkits. When the binding powers are the same, the given associativity is used to resolve the precedence. For instance, set intersection and set difference have the same binding power (30), but are both left-associative, hence $S \cap T \setminus R = (S \cap T) \setminus R$ as the left-associativeness of set intersection gives it priority over set difference. Finally, if within the same section (and all its parent sections) there are two operator templates with the same binding power (even if different kinds, say one `\function` and one `\generic`), but different associativity, a parsing error is raised since precedence cannot be decided. For instance, if we have a section with `set_toolkit` as its parent, and we define a new `\function` operator template with binding power 30 and as right associative, a parsing error is raised, as it is not possible to decide its precedence (*i.e.,* it conflicts with the operator template definition for $\cup$).

Note that generic operator templates, such as finite subsets ($\mathbb{F}\ \_$) and total functions ($\_ \rightarrow \_$), are not to be confused with a generic reference expression instantiation, such as empty sets ($\varnothing[\mathbb{N}]$), which is not given as an operator template, but rather a reference name. Moreover, when generic references are instantiated by the typechecker they are implicit ($\varnothing$), whereas when given by the user they are explicit ($\varnothing[\mathbb{N}]$—the empty set of natural numbers).

When defining operator templates, we could have single arguments (`\varg`) as in the definition of set union ($\_ \cup \_$, `\varg \cup \varg`) at `set_toolkit.tex`, or variable/list arguments (`\listarg`) as in the definition of sequence display ($\langle , , \rangle$, `\langle \listarg \rangle`) at `sequence_toolkit.tex`.

Other Z style packages allow room for a keyword `\inrel`, which could be used for changing the fixture of relations that were not defined as operator templates. For instance, suppose $R \in X \leftrightarrow Y$, $x \in X$, and $y \in Y$. As R is not an operator template, the usual way of relating x and y to R would be either "$(x, y) \in R$" or "$(x \mapsto y \in R)$". With the `\inrel` keyword, one was allowed to say "$(x\ \underline{R}\ y)$" (`(x~\inrel{R}~y)`). Nevertheless, such feature is not part of the Z Standard, hence not amenable to parsing, and thus not supported in `czt.sty`.

Additionally, we add two special "keywords" as **true** (`\true`) and **false** (`\false`) to represent boolean values at the level of the logic, rather than as predicates *true* (`$true$`) and *false* (`$false$`). This is used in the Z logic of the Z Standard. It can also be used in the definition of a boolean free type in a user toolkit. This serves to illustrate how one can make use of Z markup directives once again.

```
% AMS black board B
% \bool is already defined in czt.sty just like
% \newcommand{\bool}{\zordop{\mathbb B}}

% Note the markup directives are needed for parsing
% since they are not present in any standard toolkit.
%%Zchar \bool U-0001D539
%%Zword \true True
%%Zword \false False
\begin{zed}
   \bool ::= \false | \true
\end{zed}
```

$\mathbb{B} ::= \mathbf{false} \mid \mathbf{true}$

Apart from typesetting purposes, logic boolean values can be used, for instance, to use Z as a meta-language to specify the semantics of other languages [**?**].

5.2.2 `set_toolkit.tex`

The `set_toolkit` defines symbols for what a relation is, and operators about sets and finite sets. In Table 14, we present the list its symbols. The **Role** column contains the details for each operator template, or "*XXX name*" when the symbol is not an operator but a name, where the *XXX* determines its fixture. Infix function and generic operator templates have their binding power given as numbers, and associativity given as either LA (left-associative) or RA (right-associative). Non-infix operator templates have their type and fixture given. For ease of reference, we also add the `\varg` arguments to the LaTeX rendering column (but not the verbatim LaTeX itself for clarity).

Description	**Role**	**Rendering**	**LaTeX**
Relation space	generic 5 RA	$\_ \leftrightarrow \_$	`\rel`
Function space	generic 5 RA	$\_ \rightarrow \_$	`\fun`
Not set member	infix relation	$\_ \notin \_$	`\notin`
Inequality	infix relation	$\_ \neq \_$	`\neq`
Empty set	nofix name	$\varnothing$	`\emptyset`
Subset	infix relation	$\_ \subseteq \_$	`\subseteq`
Proper subset	infix relation	$\_ \subset \_$	`\subset`
Non-empty sets	prefix name	$\mathbb{P}_1 \_$	`\power_1`
Set union	function 30 LA	$\_ \cup \_$	`\cup`
Set intersection	function 40 LA	$\_ \cap \_$	`\cap`
Set difference	function 30 LA	$\_ \setminus \_$	`\setminus`
Set symmetric diff.	function 25 LA	$\_ \ominus \_$	`\symdiff`
Generalised union	prefix name	$\bigcup \_$	`\bigcup`
Generalised intersection	prefix name	$\bigcap \_$	`\bigcap`
Finite sets	prefix generic	$\mathbb{F} \_$	`\finset`
Non empty $\mathbb{F}$	prefix generic	$\mathbb{F}_1 \_$	`\finset_1`

Table 14: `set_toolkit.tex`symbols (A.2.5.1, B.3, B.4)

The empty set symbol within the usual LaTeX distribution (as found in file `fontmath.ltx` with font encoding `OMS/cmsy/m/n` and hex number `"3B`) is slightly different from the mathematical empty set symbol, which is present in the AMS font. Because of this, when using `czt.sty`, one can access the original empty set symbol with `\mathemptyset`, which is rendered in LaTeX as $\emptyset$.

5.2.3 `relation_toolkit.tex`

The `relation_toolkit` has `set_toolkit` as its parent and defines symbols for: maplets; domain and range; relational and functional composition; domain and range restriction and substraction; relational inversion and overriding; and transitive and reflexive transitive closures over relations. In Table 15, we present its symbols.

Description	**Role**	**Rendering**	**LaTeX**
Binary tuple projection	ordinary name	*first*	`first~\varg`
Binary tuple projection	ordinary name	*second*	`second~\varg`
Relation maplet	function 10 LA	$\_ \mapsto \_$	`\mapsto`
Domain of relation	prefix name	$\mathrm{dom}\,\_$	`\dom`
Range of relation	prefix name	$\mathrm{ran}\,\_$	`\ran`
Identity relation	prefix generic	$\mathrm{id}\,\_$	`\id`
Relational composition	function 40 LA	$\_ \fatsemi \_$	`\comp`
Functional composition	function 40 LA	$\_ \circ \_$	`\circ`
Domain restriction	function 65 LA	$\_ \lhd \_$	`\dres`
Range restriction	function 60 LA	$\_ \rhd \_$	`\rres`
Domain subtraction	function 65 LA	$\_ \ndres \_$	`\ndres`
Range subtraction	function 60 LA	$\_ \nrres \_$	`\nrres`
Relational inversion	prefix function	$\_^{\sim}$	`\inv`
Relational image left	mixfix function	$\_\llparenthesis$	`\limg`
Relational image right	mixfix function	$\_\rrparenthesis$	`\rimg`
Overriding	function 50 LA	$\_ \oplus \_$	`\oplus`
Transitive closure	posfix function	$\_^{+}$	`\plus`
Reflexive ($\_^{+}$)	posfix function	$\_^{*}$	`\star`

Table 15: `relation_toolkit.tex`symbols (A.2.5.2, B.5)

This toolkit defines tuple projection functions that do not use markup directives and are not given as operator templates, hence have no special LaTeX markup associated with them. Despite this fact, the usual LaTeX rendering is (historically) given as if they were Z keywords. To achieve this effect, however, the user need define his own "special" rendering for that markup. For instance, *first* and *second*, which project the first and second elements of a given binary tuple, are defined with ordinary names (*i.e.,* no markup directive) in `relation_toolkit.tex`. So, some users prefer to have keyword-like typesetting, which can be done as `\zkeyword{first}` (**first**). Unfortunately, this is no longer parseable, since `\zkeyword` is not part of the Z lexis, but rather a LaTeX rendering markup. Nevertheless, if the user still wants to keep a nice LaTeX rendering, she could just define the appropriate LaTeX command as an alternative markup for the name in question through markup directives. For our example, to have "*first*" typeset like a keyword (**first**), one should add the following markup directive and new LaTeX command:

```
\newcommand{\first}{\zpreop{\zkeyword{first}}}
%%Zpreword \first first
```

The markup directive will tell the parser to treat the command `\first` as the string `first`, which is loaded from `relation_toolkit.tex`. Then LaTeX can now render `\first` as desired (**first**). Furthermore, if the user wants to keep both choices conditional to the package option `tkkeyword`, one can just use the `\ztoolkit{first}`, instead. With current option, it typesets as first. Finally, note that, since **first** a prefix word, we also wrap it with a `\zpreop`. This way proper spacing is added, and one does not need to typeset it as `$\first~x$` (**first** x) but just `$\first x$` (**first** x), as opposed to the mandatory hard space in `$first~x$` (*first* x) to avoid the wrong spacing from `$first x$` (*firstx*).

The Z Standard also leaves room for mixfix (mixed fixture) operator templates, although those are more rarely used. One such operator is used for the definition of relational imagine as

```
%%Zinchar   \limg U+2987
%%Zpostchar \rimg U+2988
\begin{zed}
\function (\_ \limg \_ \rimg)
\end{zed}
```

So, each bracketing symbol is treated with a different fixture. That is, ⦇ is treated as an infix operator, whereas ⦈ is treated as a posfixed one. This combination makes the relational image mixfix operator template as defined above.

The AMS/Lucida bright font(s) already define(s) the `\star` symbol as "$\star$" (*e.g.,* `msam10`, hex-number `"46`), rather than the "*" we want. Because of this, when using `czt.sty`, one can access the original AMS/Lucida bright star symbol with the `\mathstar` command, which is rendered as "$\star$".

For relational inverse (`R\inv`), the Z Standard does not specify it with superscripting word glues [**?**, A.2.4.3]. Thus, its rendering is "$R\,^{\sim}$", and it should not be superscripted as in "$R^{\sim}$", despite this being more common. This may perhaps be a Z Standard typo.

5.2.4 `function_toolkit.tex`

The `function_toolkit` has `relation_toolkit` as its parent and defines symbols for generic operator templates representing the various subsets of function spaces, and a few relational predicates for sets. In Table 16, we present its list of symbols. Lucida Bright fonts render some of these symbols differently, if (and when) loaded.

Disjointness of a relation states that a set of sets has no overlapping elements (*i.e.,* their pairwise set intersection is empty), whereas partitioning represents a disjoint set of sets that covers the whole elements of the set's type (*i.e.,* the generalised union of all sets being disjoint represents the whole type).

5.2.5 `number_toolkit.tex`

The `number_toolkit` defines symbols for integer arithmetic. In Table 17, we present its list symbols. Note that summation is defined as an operator template in `prelude.tex`, but most of its properties are defined in `number_toolkit.tex`,

Description	**Role**	**Rendering**	**LaTeX**
Partial function	generic 5 RA	$\_ \pfun \_$	`\pfun`
Partial injection	generic 5 RA	$\_ \pinj \_$	`\pinj`
Injection	generic 5 RA	$\_ \rightarrowtail \_$	`\inj`
Partial surjection	generic 5 RA	$\_ \psurj \_$	`\psurj`
Surjection	generic 5 RA	$\_ \twoheadrightarrow \_$	`\surj`
Bijection	generic 5 RA	$\_ \bij \_$	`\bij`
Finite partial function	generic 5 RA	$\_ \ffun \_$	`\ffun`
Finite partial injection	generic 5 RA	$\_ \finj \_$	`\finj`
Disjoint sets	prefix relation	$\mathrm{disjoint}\,\_$	`\disjoint`
Set partitioning	infix relation	$\_ \mathrm{partition} \_$	`\partition`

Table 16: `function_toolkit.tex`symbols (A.2.5.3, B.6)

Description	**Role**	**Rendering**	**LaTeX**
$\mathbb{N}$ successor function	prefix function	$succ\_$	`succ \varg`
Integers	ordinary name	$\mathbb{Z}$	`\num`
Arithmetic negation	prefix function	$-\_$	`\negate`
Subtraction	function 30 LA	$\_ - \_$	`-`
Summation	function 30 LA	$\_ + \_$	`+`
Less-than equal-to	infix relation	$\_ \leq \_$	`\leq`
Less-than	infix relation	$\_ < \_$	`<`
Greater-than equal-to	infix relation	$\_ \geq \_$	`\geq`
Greater-than	infix relation	$\_ > \_$	`>`
Non empty $\mathbb{N}$	prefix name	$\mathbb{N}_1$	`\nat_1`
Non empty $\mathbb{Z}$	prefix name	$\mathbb{Z}_1$	`\num_1`
Multiplication	function 40 LA	$\_ * \_$	`*`
Integer division	function 40 LA	$\_ \mathrm{div} \_$	`\div`
Integer modulus	function 40 LA	$\_ \mathrm{mod} \_$	`\mod`

Table 17: `number_toolkit.tex`symbols (A.2.5.4, B.7)

hence we left it here. Subtraction is defined in terms of summation and unary negation (*e.g.,* $-\_$, `\negate`).

Like what happened in `relation_toolkit.tex`, where definitions were given without markup directives, in `number_toolkit.tex`, the successor function for natural numbers (*succ*) is also defined without markup directives, yet one may be familiar with its specialised rendering as succ (`\ztoolkit{succ}`). This is slightly different from *first* and *second* from `relation_toolkit.tex`, as *succ* is defined as an operator template in `number_toolkit.tex`, hence the `\varg` on its description in Table 17.

The division symbol within the usual LaTeX distribution (`fontmath.ltx` with font encoding `OMS/cmsy/m/n` and hex value `"04`) is different from the Z integer division symbol, which is given as a Z toolkit word (`\ztoolkit{div}`) in `czt.sty`. To access the original definition, one should use `\mathdiv` ($\div$), instead.

5.2.6 `sequence_toolkit.tex`

The `sequence_toolkit` has `function_toolkit` and `number_toolkit` as its parents and defines range, relational iteration, set cardinality, min/max, and finite sequences and its operators. In Table 18, we present its list of symbols.

Description	**Role**	**Rendering**	**LaTeX**
Number range	function 20 LA	$\_ \mathrel{..} \_$	`\upto`
Iteration	ordinary name	$\mathit{iter}\ \_\ \_$	`iter`
Iteration	prefix function	$(\_^{\,-})$	`\varg~^{~\varg~}`
$\mathbb{F}$ cardinality	prefix function	$\#\,\_$	`\#`
Minimum	prefix function	$\mathit{min}\_$	`min~\varg`
Maximum	prefix function	$\mathit{max}\_$	`max~\varg`
Finite seq.	prefix generic	$\mathrm{seq}\,\_$	`\seq`
Non empty seq	prefix name	$\mathrm{seq}_1\,\_$	`\seq_1`
Injective seq.	prefix generic	$\mathrm{iseq}\,\_$	`\iseq`
Seq. brackets	mixfix function	$\langle,,\rangle$	`\langle \listarg \rangle`
Concatenation	function 30 LA	$\_ \frown \_$	`\cat`
Seq. reverse	ordinary name	$\mathit{rev}\_$	`rev~\varg`
Seq. head	ordinary name	$\mathit{head}\_$	`head~\varg`
Seq. last	ordinary name	$\mathit{last}\_$	`last~\varg`
Seq. tail	ordinary name	$\mathit{tail}\_$	`tail~\varg`
Seq. front	ordinary name	$\mathit{front}\_$	`front~\varg`
Seq. re-indexing	ordinary name	$\mathit{squash}\_$	`squash~\varg`
Seq. extraction	function 45 LA	$\_ \upharpoonleft \_$	`\extract`
Seq. filtering	function 40 LA	$\_ \upharpoonright \_$	`\filter`
Seq. prefix	prefix relation	$\_\ \mathrm{prefix}\ \_$	`\prefix`
Seq. suffix	prefix relation	$\_\ \mathrm{suffix}\ \_$	`\suffix`
Seq. infix	prefix relation	$\_\ \mathrm{infix}\ \_$	`\infix`
Dist. concat.	ordinary name	${}^\frown/$	`\dcat`

Table 18: `sequence_toolkit.tex`symbols (A.2.5.5, B.8)

In `sequence_toolkit.tex`, few ordinary names or operator templates without markup directive also are typeset as keywords. They are: relation iteration (iter $R\,i$) and its superscript version ($R^{\,i}$); minimum (*min*) and maximum (*max*) of a set of numbers; sequence *rev*erse, *head*, *last*, *tail*, *front*, and *squash*; and distributed concatenation (${}^\frown/$). It is questionable if some of them should be made prefix function operator templates in the Z Standard. Note that, as these are ordinary names, no special LaTeX spacing scheme is in place. Thus, although not explicitly required by the CZT tools, to properly render these names, a hard space is required in order to separate them from their arguments (*e.g.,* "*rev s*", `$rev~s$`). Otherwise, LaTeX will typeset them as a single word (*e.g.,* "*revs*", `$rev s$`). Again, if wanted, markup directives with corresponding LaTeX macros as `\ztoolkit` can be added.

5.2.7 `standard_toolkit.tex`

The `standard_toolkit` has `sequence_toolkit` as its parent and defines nothing. It is the Z section implicitly inherited if no **section** keyword is present within a given file. Such files have so-called "implicit" sections, where the implicit section is named as the file (without its extension), where the `standard _toolkit` is its parent [**?**, B.9].

6 Z-LaTeX environments

In Table 19, we describe all the Z-LaTeX environments used to typeset the various Z paragraphs, such as: Z section headers containing the section name and its (optional, possibly empty, list of) parents; horizontal paragraphs like given sets, operator templates, free types, horizontal schemas, and unnamed conjectures; named conjecture paragraphs; axiomatic and generic axiomatic definitions; and schema and generic schema definitions. In many of these paragraphs, the `\where` keyword is used to separate the declaration part from the predicate part. The `ENDCHAR` is used to mark the end of all Z paragraphs within the Unicode character stream.

Description	**Markup**	**LaTeX**
Section header	`ZEDCHAR`	`\begin{zsection}`
Horizonal paragraph	`ZEDCHAR`	`\begin{zed}`
Named conjecture	`ZEDCHAR`	`\begin{theorem}{thm}`
Axiomatic definition	`AXCHAR`	`\begin{axdef}`
Generic axdef	`AXCHAR GENCHAR`	`\begin{gendef}`
Schema definition	`SCHCHAR`	`\begin{schema}{S}`
Generic schema	`SCHCHAR GENCHAR`	`\begin{schema}{S}[X]`
Declaration separator	`~|~`, or `~\mid~`	`\where`
End of all Z paras	`ENDCHAR`	`\end{XXX}`

Table 19: Z-LaTeX environments (A.2.6, A.2.7)

Only material within Z paragraphs and LaTeX markup directives are treated by CZT tools as part of a formal Z specification. Insofar as tools are concerned, everything else (*e.g.,* plain text, LaTeX comments, other LaTeX commands, *etc.*) is treated as a Z narrative paragraph, which can contain arbitrary text.

To illustrate these boxes, we introduce a few Z paragraphs below. They are inspired in Mike Spivey's guide to Z-LaTeX markup (*i.e.,* `zed2e.tex`). Firstly, we add a series of horizonal paragraphs.

```
\begin{zed}
   % Hard spaces (~) are optional below. They were
   % added for (personal) aesthetic reasons.
   [Set]
   \also    % small vertical space
   List ~~::=~~ leaf | const \ldata List \rdata \\
   Sch  ~~==~~ [~ x, y: \nat | x > y ~] \\
   Sch2 ~~==~~ Sch \land [~ z: \num ~]
\end{zed}
```

$[\mathit{Set}]$

$\mathit{List} ::= \mathit{leaf} \mid \mathit{const}\langle\!\langle \mathit{List} \rangle\!\rangle$
$\mathit{Sch} \;==\; [\, x, y : \mathbb{N} \mid x > y \,]$
$\mathit{Sch2} \;==\; \mathit{Sch} \land [\, z : \mathbb{Z} \,]$

Next, we typeset an axiomatic definition.

```
\begin{axdef}
   f, g: \power~\nat \fun (\num \cross \seq~\arithmos)
\where
\zbreak     % may not break, depends on page placement
   \forall S, T: \power~\nat | f~S \subseteq g~S @ \\
        \t1 first~(f~S) < \#~(g~S).2
\end{axdef}
```

$f, g : \mathbb{P}\, \mathbb{N} \to (\mathbb{Z} \times \mathrm{seq}\; \mathbb{A})$

$\forall\, S, T : \mathbb{P}\, \mathbb{N} \mid f\; S \subseteq g\; S \bullet$
$\quad \mathit{first}\,(f\; S) < \#\,(g\; S).2$

After that, we have a simple vertical schema.

```
\begin{schema}{Test}
   x, y: \nat; S, T: \power_1~\nat
\where
   x > y \\
   S \subset T \\
\znewpage           % certainly breaks
   x \neq y = 0
   \Also            % medium vspace
   x \in S \land y \notin T
\end{schema}
```

Test

$x, y : \mathbb{N};\ S, T : \mathbb{P}_1\ \mathbb{N}$

$x > y$
$S \subset T$

$x \neq y = 0$

$x \in S \land y \notin T$

Below we typeset a generic axiomatic definition.

```
\begin{gendef}[X, Y]
   S, T: \power~(X \cross Y)
\where
   S \subseteq T
   \ALSO           % big vertical space

   \exists U: \power~(X \cross Y) \spot \\
        \t2 U \subset (S \cup T)
\end{gendef}
```

$[X, Y]$

$S, T : \mathbb{P}\,(X \times Y)$

$S \subseteq T$

$\exists\, U : \mathbb{P}\,(X \times Y) \bullet U \subset (S \cup T)$

And finally, a generic schema.

```
\begin{schema}{GenTest}[X]
   a: X; b: \power~X
\where
   a \in b
\end{schema}
```

$\mathit{GenTest}[X]$

$a : X;\; b : \mathbb{P}\; X$

$a \in b$

For schemas without names, which are not recognised by the parser, one could use the `\begin{plainschema}` environment.

```
\begin{plainschema}
    x, y: \nat
 \where
    x = y
\end{plainschema}
```

$$x, y : \mathbb{N}$$

$$x = y$$

Finally, stared versions of the usual Z environments can be used to typeset Z-LaTeX, but having its text ignored by the CZT tools as a narrative paragraph.

```
\begin{zed*}
   [NotParsed]
\end{zed*}
\begin{axdef*}
   a : \arithmos
\end{axdef*}
\begin{gendef*}[X]
   x: X
\end{gendef*}
\begin{schema*}{NotParsed}[X]
   x, y: X
\where
   x > y
\end{schema*}
```

$$[NotParsed]$$

$$a : \mathbb{A}$$

$$[X]$$

$$x : X$$

$$NotParsed[X]$$

$$x, y : X$$

$$x > y$$

After we have done that, let us test trailing spaces after Z paragraph environments are not affecting L/R mode indentation spacing, a known problem in some old Z-LaTeX style files. Say, let us define a new operator template as a prefix function. For that we also add, together with the operator definition, its (Z) LaTeX markup directive and associated LaTeX markup command as a `\ztoolkit`.

```
% Unicode markup in markup directive is the "text" to use
\newcommand{\test}{\ztoolkit{test}}

\begin{zed}
   %%Zword \test test
   \function (\test \_)
\end{zed}
Now some text to see if paragraph mode indentation is right.
\[ \forall x: \nat_1 @ x > 0 \]
What about with math display environments?
All seems okay.
```

$\mathbf{function}(\mathrm{test}\_)$

Now some text to see if paragraph mode indentation is right.

$$\forall x : \mathbb{N}_1 \bullet x > 0$$

What about with math display environments? All seems okay. Finally, let us test the named conjecture environment.

```
\begin{theorem}{Thm1}
   \forall x: \nat @ x \geq 0
\end{theorem}
```

$\mathbf{theorem}\ \mathrm{Thm1}$
$\quad \forall x : \mathbb{N} \bullet x \geq 0$

Unfortunately, I could not find a way to make named conjectures colourful, whenever colour is enabled in Z math mode.

7 Controlling definition counting

One interesting new feature added to the style file is the ability to (**roughly**) count the number of each Z definition specified in a given portion of LaTeX. The counting is controlled by few package options, which determine if counting should be globally, by Chapter, or by Section. The actual counters can also be directly accessed and/or changed, where automatic counting can also be switched on and off if one wants tighter control over when what should be counted.

There are three package options associated with counting:

- `cntglobally`: counts definitions globally;
- `cntbychapter`: definition count reset by each Chapter;
- `cntbysection`: definition count reset by each Section.

For the local counters by Chapter and Section we also have a global counter. This way one can keep track of both local and global amounts. Another important command is the boolean flag `\CountDefs`. It determines whether counting switched on (`\CountDefstrue`) or off (`CountDefsfalse`). It can be used to selectively count parts of a specification.

Another command to use is `\ZDeclSummary`. It typesets a table with seven lines and two or three columns, depending on the counting option chosen. The lines are: the column header; the number of unboxed items, which represent given sets, free types, (named) conjectures, operator templates, and abbreviations (*e.g.,* constant definitions or horizontal schemas); the number of axiomatic and generic axiomatic definitions; the number of schemas and generic schemas; and finally the total number of declarations. The columns are: each line description; and the totals per line, where local totals are added in the case of section and chapter options had been chosen. Moreover, each number given in this table is represented internally by a LaTeX counter, which the user can both access and interfere with if needed.

For example, for this file, where we have chosen the counting by section option, issuing `\ZDeclSummary` produces the contents shown in Table 20. Note

Z Declarations	**This Section**	**Globally**
Unboxed items	0	11
Axiomatic definitions	0	2
Generic axiomatic defs.	0	2
Schemas	0	1
Generic schemas	0	1
Theorems	0	1
Proofs	0	0
Total	**0**	**18**
Unknown?	**0**	**0**

Table 20: Summary of Z declarations for Section 7.

that, since this section has no specification, nothing is counted locally. Moreover, to refer to the table `\ZDeclSummary` typesets, a unique label is specified for it, and one can use the command `\ref{tbl:zdecl:\thesection}`. For Chapter counting, one needs to use `\thechapter` in the place of `\thesection`, and just `\ref{tbl:zdecl:global}` when counting globally. These labels `should` not be defined by users, or a *multiply defined label* LaTeX warning will arise.

More advanced usage of the counting facility can be achieved by directly influencing the way the counters work, either by accessing their value, or interfering in the way counting works. One should not do that usually, unless extending the counting facilities to include other pieces of information, say, number of examples or proofs given by a specification.

Declarations	**Local counter name**	**Global counter name**
Unboxed items	`cntZunboxed`	`cntZtotunboxed`
Axiomatic definitions	`cntZaxdef`	`cntZtotaxdef`
Generic axiomatic defs.	`cntZgendef`	`cntZtotgendef`
Schemas	`cntZschema`	`cntZtotschema`
Generic schemas	`cntZgenschema`	`cntZtotgenschema`
Total	`cntZdecl`	`cntZtotdecl`

Table 21: Z definition counters

When counting locally by Chapter or Section, the local counter names are reset according to LaTeX's Chapter or Section counters change. The equivalent global counters are not reset and serve to accumulate the global totals, also included in the information. Note that the local total counter (`cntZtotdecl`) simply sums the local counters, whereas the global total counter (`cntZdecl`) sums the global counters. Finally, when counting globally, only the global counters are.

One can access the value of each of these counters using the usual LaTeX commands to manipulate counters, such as: `\stepcounter{name}`, which increases counter `name` by one; and `\thename`, which accesses the `\arabic{name}` counter value. Moreover, when counting by Chapter or Section, we redefine the contents of `\thename` value for all local counters to include the Chapter or Section where the local counter belongs to. For example, for the value of `cntZunboxed`, the result of `\thecntZunboxed` is "0 unboxed items in Section 7". One can easily redefine such command for whatever else is appropriated.

Finally, there are two more low-level LaTeX commands used to control counting. The command `\@countZpara` is called within the various Z-LaTeX environments to actually perform the counters update. A single command within all environment simplify the counting code. If `\CountDefs` is true, then when `\@countZpara` is expanded, the appropriate LaTeX counters described above are updated depending on the value of a low-level TeX register `\@zcountingwhat`. That is, the TeX register determines what is being counted. The values used are:

- `\@zcountingwhat=0`: counts unboxed items affecting `cntZunboxed`;
- `\@zcountingwhat=1`: counts axiomatic defs. affecting `cntZaxdef`;

- `\@zcountingwhat=2`: counts generic axiomatic defs. affecting `cntZgendef`;
- `\@zcountingwhat=3`: counts schemas affecting `cntZschema`;
- `\@zcountingwhat=4`: counts generic schemas affecting `cntZgenschema`;
- `\@zcountingwhat=99`: counts nothing affecting no counter.

One use for `\@zcountingwhat=99` is, for instance, to avoid counting non-parseable definitions, such as the stared versions of the Z-LaTeX environments (*e.g.,* `\begin{zed*}`). Package developers extending this counting facility can use this TeX register and other low-level commands to count their own definitions. When extending Z-LaTeX environments, package developers are advised to look into the more detailed documentation of `czt.sty`, which explains more of the low-level LaTeX/TeX commands available.

Axiomatic and generic definitions are counted within each `\begin{axdef}` and `\begin{gendef}` commands, whereas schemas and generic definitions are counted within each `\begin{schema}{name}` and `\begin{schema}[X]{name}`, respectively. Unboxed items are introduced through the `\begin{zed}` or `\begin{theorem}{name}` commands. Nevertheless, counting must also take place when Z-LaTeX new lines are introduced, as they represent declaration separators. That is, every `\\` or `\also` command within `\begin{zed}` will also increase the counter. The same applies for the variations of the `\also` command (*i.e.,* `\Also` and `\ALSO`). Note that, since the Z Standard allows multiple new lines to separate commands (*i.e.,* more than one `\\` between commands), this could create a discrepancy between the number of definitions counted, and the actual number specified. Such discrepancy may only occur for unboxed items, whenever their specification has multiple line separators.

8 Extra macros and commands from `czt.sty`

There are a few extra macros the user may refer to when extending the `czt.sty`, or adding her own markup directives. They are summarised in Table 22. File version, date, and description are simple strings with information about `czt.sty`. The various operator wrappers are used to tell LaTeX how spaces for some particular markup should be treated. They follow the usual LaTeX mathematical operators spacing rules (see [?, p. 525, Table 8.7]). Some symbols can be increased or decreased relative to their base symbol. For instance, the symbol for schema composition ($\fatsemi$) is the `\zBig` version of the symbol for relational composition ($\fatsemi$). Similarly, partial function spaces ($\pfun$) are just the `\p` version of total functions ($\rightarrow$). Finally, block alignment can be used so that the treatment of new line within the block adds extra spacing just after the new line.

9 Conclusions and acknowledgements

In this document, we presented a guide to typesetting ISO Standard Z [?] in LaTeX when typeset using the `czt.sty`. The document is divided to mirror the

Description	**LaTeX**
`czt.sty` version	`\fileversion`
`czt.sty` date	`\filedate`
`czt.sty` description	`\filedesc`
`czt.sty` file name	`\cztstylefile`
Prefix operators	`\zpreop{XXX}`
Posfix operators	`\zpostop{XXX}`
Binary operators	`\zbinop{XXX}`
Relational operators	`\zrelop{XXX}`
Ordinary operators	`\zordop{XXX}`
Big symbol	`\zbig{XXX}`
Bigger symbol	`\zBig{XXX}`
Even bigger symbol	`\zBIG{XXX}`
Smaller symbol	`\zSmall{XXX}`
Even smaller symbol	`\zsmall{XXX}`
Partial symbol	`\p{XXX}`
Finite symbol	`\f{XXX}`
Block alignment env.	`\begin{zblock}\end{zblock}`

Table 22: Extra LaTeX macros in (`czt.sty`)

Standard as much as possible. This style file is the result of merging, filtering, and removing definitions from various other style files, such as `oz.sty`, `soz.sty`, `zed-csp.sty`, `zed.sty`, `fuzz.sty`, `z-eves.sty`, and so on.

The main design decision behind this document follows CZT guideline that "what you type is what you model". That is, the document "as-is" becomes the source Standard Z (LaTeX) specification to be processed by tools. Other design decisions included: i) keep the style file as minimal, simple, and consistent as possible; ii) document and acknowledge macro definition choices and their origin (when different); iii) normalise definitions for consistency; iv) complete missing cases with either normative rules from the Standard or using common sense; v) keep the style file well documented, but not verbose; and vi) follow order of definitions from Z Standard document.

As the `czt.sty` may be used by both language extensions and LaTeX users, we also provided and explained a series of useful macros for LaTeX rendering that bare no relation with the Standard or the tools. They are useful for LaTeX typesetting only, and are explained in Section 1.2, and Section 8.

We tried to present, as exhaustively as possible, the use of every one of such commands with LaTeX markup typeset in verbatim mode for clarity and reference. We summarise them all in an Appendix below. More details can be found at the `czt.dvi` file generated with the `docstrip` utility on the `czt.dtx` document from the CZT distribution.

Finally, the author would like to thank *QinetiQ Malvern* in the UK for its long term support for the development of formal verification tools here at York. Also, the work to prepare this document and its companion style file benefited immensely by the good work of previous package builders for Z, namely Sebastian Rahtz (Object Z, `oz.sty`), Mike Spivey (ZRM and Fuzz, `zed.sty, fuzz.sty`), Jim Davies (ZRM and CSP$_M$, `zed-csp.sty`), Ian Toyn (Standard Z Editor, `ltcadiz.sty, soz.sty`), and Mark Utting (original CZT style based on `oz.sty`, `czt.sty`). Moreover, I would like to thank all the people in the `czt-devel` mailing list for their helpful comments on my many questions. Finally, I need to thank my York colleagues Jim Woodcock and Juan Perna for many helpful discussions about tool design and LaTeX typesetting.

10 Features left out

There were several features left out from the various packages we got inspiration from which might be of good use in typesetting LaTeX specifications, as shown below in Table 23.

Description	**Source**	**LaTeX**
Multiple column math mode	`oz.sty`	`\begin{sidebyside}`
Comment in math mode	`oz.sty`	`\comment{XXX}`
indented new lines alignment	`oz.sty`	various
Tabular alignment math mode	`zed.sty`	`\begin{syntax}`
Hand written proofs	`zed.sty`	`\begin{argue}`
Inference rules	`zed.sty`	`\begin{infrule}`
Mechanical proof scripts	`z-eves.sty`	`\begin{zproof}`
Labelled predicates	`z-eves.sty`	`\Label{XXX}`
Various new line alignment	`z-eves.sty`	`\+, \-, \\`

Table 23: Some LaTeX macros left out from other style files

Although some of them could be introduced without problem as *e.g.,*

```
\begin{sidebyside}...\end{sidebyside}
```

for most others the trouble is their presence within the Z-LaTeX lexis. That is, their presence would be detected by the parser as an error, hence they were left out.

Finally, note that the Z Standard does not define a toolkit for multi sets also known as bags. That is despite the fact most Z tools do, and the symbols are well known from Spivey's guide [**?**]. Eventually, we should have in CZT extra toolkits from either known sources and rigorous experiments.

A Reference card

A.1 Letters

Special Greek

Δ	\Delta
Ξ	\Xi
θ	\theta
λ	\lambda
μ	\mu
Φ	\Phi

Small Greek

α	\alpha
β	\beta
γ	\gamma
δ	\delta
ϵ	\epsilon
ζ	\zeta
η	\eta
ι	\iota
κ	\kappa
ν	\nu
ξ	\xi
π	\pi
ρ	\rho
σ	\sigma
τ	\tau
υ	\upsilon
ϕ	\phi
χ	\chi
ψ	\psi
ω	\omega

Capital Greek

Γ	\Gamma
Θ	\Theta
Λ	\Lambda
Π	\Pi
Σ	\Sigma
Υ	\Upsilon
Φ	\Phi
Ψ	\Psi
Ω	\Omega

A.2 Special Z characters

Stroke chars

$'$	'
!	!
?	?

Brackets

(	(
)	)
[	[
]	]
{␣	\{~
␣}	~\}
⦉	\lblot
⦊	\rblot
⟪	\ldata
⟫	\rdata
⇒ _ ⇐	_
⇒ _ ⇐	\varg

Spacing

⇒⇐	~
⇒ ⇐	\␣
⇒⇐	\,
⇒ ⇐	\:
⇒ ⇐	\;

$\Rightarrow \quad \Leftarrow$	`\t1`
$\Rightarrow \qquad \Leftarrow$	`\t2`
new line	`\\`
small vspace	`\also`
med. vspace	`\Also`
big. vspace	`\ALSO`
small break	`\zbreak`
med. break	`\zBreak`
big. break	`\ZBREAK`
new page	`\znewpage`

A.3 Z Notation

Logic

$\vdash P$	`\vdash P`
$P \land Q$	`P \land Q`
$P \lor Q$	`P \lor Q`
$P \Rightarrow Q$	`P \implies Q`
$P \Leftrightarrow Q$	`P \iff Q`
$\lnot P$	`\lnot P`
$\forall x : T \bullet P$	`\forall x: T @ P`
$\exists x : T \bullet P$	`\exists x: T @ P`
$\exists_1 x : T \bullet P$	`\exists_1 x: T @ P`
$x \in S$	`x \in S`
$X \times Y$	`X \cross Y`
$S \mathbin{\Big\backslash} (x)$	`S \hide (x)`
$S \upharpoonright T$	`S \project T`
$S \fatsemi T$	`S \semi T`
$S \gg T$	`S \pipe T`
$E ⦂ T$	`E \typecolon T`
true	`\true`
false	`\false`
$\mathbb{B}$	`\bool`

Z keywords

section *name*	`\SECTION name`
parents $s1, s2$	`\parents s1, s2`
if P **then** $E1$ **else** $E2$	`\IF P \THEN E1 \ELSE E2`
let $x == y \bullet P$	`\LET x == y @ P`
pre S	`\pre S`
function	`\function`
relation	`\relation`
generic	`\generic`
leftassoc	`\leftassoc`
rightassoc	`\rightassoc`

A.4 Mathematical toolkits

Set toolkit

$X \leftrightarrow Y$	`X \rel Y`
$X \rightarrow Y$	`X \fun Y`
$x \notin S$	`x \notin S`
$x \neq y$	`x \neq y`
$\varnothing$	`\emptyset`
$S \subseteq T$	`S \subseteq T`
$S \subset T$	`S \subset T`
$\mathbb{P} X$	`\power X`
$\mathbb{P}_1 X$	`\power_1 X`
$S \cup T$	`S \cup T`
$S \cap T$	`S \cap T`
$S \setminus T$	`S \setminus T`
$S \ominus T$	`S \symdiff T`
$\bigcup SS$	`\bigcup SS`
$\bigcap SS$	`\bigcap SS`
$\mathbb{F} X$	`\finset X`
$\mathbb{F}_1 X$	`\finset_1 X`

Relation toolkit

first t	`first~t`
second t	`second~t`
$x \mapsto y$	`\mapsto`
$\mathrm{dom}\, R$	`\dom`
$\mathrm{ran}\, R$	`\ran`
$\mathrm{id}\, R$	`\id`

$R \mathbin{⨾} S$	`R \comp S`
$R \circ S$	`R \circ S`
$R \lhd S$	`R \dres S`
$R \rhd S$	`R \rres S`
$R \mathbin{⩤} S$	`R \ndres S`
$R \mathbin{⩥} S$	`R \nrres S`
$R^{\sim}$	`R \inv`
$R \llparenthesis S \rrparenthesis$	`R \limg S \rimg`
$R \rrparenthesis$	`R \rimg`
$R \oplus S$	`R \oplus S`
R^{+}	`R \plus`
R^{*}	`R \star`

Function toolkit

$X \mathrel{⇸} Y$	`X \pfun Y`
$X \mathrel{⤔} Y$	`X \pinj Y`
$X \rightarrowtail Y$	`X \inj Y`
$X \mathrel{⤀} Y$	`X \psurj Y`
$X \twoheadrightarrow Y$	`X \surj Y`
$X \mathrel{⤖} Y$	`X \bij Y`
$X \mathrel{⇻} Y$	`X \ffun Y`
$X \mathrel{⤕} Y$	`X \finj Y`
$\operatorname{disjoint} S$	`\disjoint S`
$S \mathrel{\text{partition}} T$	`S \partition T`

Number toolkit

$\mathbb{A}$	`\arithmos`
$\mathbb{Z}$	`\num`
$\mathbb{Z}_1$	`\num_1`
$\mathbb{N}$	`\nat`
$\mathbb{N}_1$	`\nat_1`
$\mathbb{Q}$	`\rat`
$\mathbb{R}$	`\real`
$\mathit{succ}\ n$	`succ~n`
$-x$	`\negate x`
$x - y$	`x - y`
$x + y$	`x + y`
$x \leq y$	`x \leq y`
$x < y$	`x < y`
$x \geq y$	`x \geq y`
$x > y$	`x > y`
$x * y$	`x * y`
$x \operatorname{div} y$	`x \div y`
$x \bmod y$	`x \mod y`

Sequence toolkit

$x \mathrel{..} y$	`x \upto y`
$\mathit{iter}\ R\ i$	`iter~R~i`
$(R^{\,i})$	`R~^{~i~}`
$\#\ S$	`\#~S`
$\mathit{min}\ S$	`min~S`
$\mathit{max}\ S$	`max~S`
$\operatorname{seq} X$	`\seq X`
$\operatorname{seq}_1 X$	`\seq_1 X`
$\operatorname{iseq} X$	`\iseq X`
$\langle x, y \rangle$	`\langle x, y \rangle`
$s \frown t$	`\cat`
$\mathit{rev}\ s$	`rev~s`
$\mathit{head}\ s$	`head~s`
$\mathit{last}\ s$	`last~s`
$\mathit{tail}\ s$	`tail~s`
$\mathit{front}\ s$	`front~s`
$\mathit{squash}\ s$	`squash~s`
$S \upharpoonleft s$	`S \extract s`
$s \upharpoonright S$	`s \filter S`
$s \mathrel{\text{prefix}} t$	`s \prefix t`
$s \mathrel{\text{suffix}} t$	`s \suffix t`
$s \mathrel{\text{infix}} t$	`s \infix t`
${\frown}/\ ss$	`\dcat~ss`