

The `linguexx` package

Numbered examples, interlinear glosses and grammaticality judgments
for linguistics

Gerhard Schaden
`gerhard.schaden@univ-lille.fr`

Version 1.2 — July 31, 2026

Abstract

`linguexx` typesets the standard apparatus of linguistic writing: numbered examples with lettered sub-examples, grammaticality judgments that hang in the margin, interlinear morpheme-by-morpheme glosses with any number of tiers, cross-references that follow the numbering automatically, and examples inside footnotes. Its input syntax is deliberately terse — `\ex.` starts an example, a blank line ends it — so that data does not disappear under markup.

That syntax comes from Wolfgang Sternefeld’s `linguex`, which this package owes almost everything to, and which it is designed to replace. With one exception, any document that compiles under `linguex` should also compile under `linguexx` – and given one switch, it should look the same.

`linguexx` is a reimplementaion from the ground up, undertaken for two reasons. The first is to repair the structural weaknesses of the original. The second is to bring the glossing closer to the level of John Frampton’s `expex`: glosses may now have any number of tiers, each with its own font, rather than the two or three that `cgloss4e` allowed.

A further new feature is that the package tags the whole example apparatus, from numbered examples to interlinear glosses, for a PDF/UA-2-conformant result (checked with `veraPDF`).

The package’s only dependencies are `graphicx` and `tikz`, plus the `expl3` layer, part of the `LATEX` kernel since 2020, and it works identically under `pdfLATEX`, `XYLATEX` and `LuaLATEX`. This manual is self-contained and assumes no acquaintance with `linguex` or any other example package.

Contents

1	A quick tour & recommended usage	2
2	Origins, acknowledgments, and relation to other packages	4
3	Examples	5
3.1	Starting and ending an example	5
3.2	Sub-examples	5
3.3	Returning to a higher level: <code>\z.</code>	6
3.4	Custom labels	6
3.5	An environment interface	7
3.6	Examples in footnotes	8

4	Grammaticality judgments	8
4.1	Automatic marks	8
4.2	Space for judgments	9
4.3	Arbitrary marks: <code>\jdg</code>	9
5	Interlinear glosses	10
5.1	Two and three tiers	10
5.2	Gloss abbreviations: <code>\lpzg</code>	10
5.3	The list of abbreviations used: <code>\lpzglst</code>	11
5.4	Checking the abbreviations: <code>\lpzgcheck</code>	12
5.5	Any number of tiers: <code>\gl ... \endgl</code>	13
5.6	Fonts and spacing	13
5.7	Aligning past brackets and judgment marks	13
5.8	The language of a tier (tagged PDFs)	14
5.9	Abbreviations: <code>\exg.</code> and <code>\ag.</code>	15
6	Cross-references	16
6.1	Labels	16
6.2	References without parentheses	16
6.3	Relative references	16
6.4	Ranges	17
7	Further apparatus	18
7.1	Stacked alternatives: <code>\altn</code>	18
7.2	Glossed alternatives: <code>\altg</code>	19
7.3	Source attributions: <code>\exsource</code>	20
8	Example and glossing layout	20
9	PDF tagging and accessibility	22
9.1	Spoken forms for tagged PDFs	23
9.2	Gloss abbreviations and their expansions	23
9.3	Transliteration and the engine you compile with	24
10	Notes and limitations	25
11	Command reference	27

1 A quick tour & recommended usage

This section illustrates the recommended use for a total novice in linguistic glossing. If you already know `linguex`, you can do what you are used to do.

Load the package, with the recommended option `phantomalign` (see Section 5.7):

```
| \usepackage[phantomalign]{linguexx}
```

An example is introduced by `\ex.` — note the period, which is part of the command name — and ended by a `\z.`, or alternatively, a blank line:

```
| \ex. Colourless green ideas sleep furiously.
| \z.
```

renders as:

- (1) Colourless green ideas sleep furiously.

Numbering is automatic and runs through the document. Sub-examples are introduced by `\a.`, and continued by `\b.`, `\c.` and so on:

```
\ex.  
\a. Ich habe geschlafen.  
\b. Ich bin geschlafen.  
\z.
```

renders as:

- (2) a. Ich habe geschlafen.
b. Ich bin geschlafen.

A grammaticality judgment typed at the start of an example is recognized automatically and set in the margin, so that the example texts stay aligned:

```
\ex.  
\a. Ich habe geschlafen.  
\b. *Ich habe gestorben.  
\z.
```

renders as:

- (3) a. Ich habe geschlafen.
b. *Ich habe gestorben.

Interlinear glosses are written with `\gll` (two tiers), with the free translation on a `\glt` line. The tiers are aligned word by word, and category labels are set in small capitals with `\lpzg`:

```
\ex. \gll Ich schlief. \\  
      I    sleep.\lpzg{pst} \\  
\glt `I slept.'  
\z.
```

renders as:

- (4) Ich schlief.
I sleep.PST
'I slept.'

To refer to an example, label it and use `\ref`, exactly as with any other $\text{\LaTeX}$ counter; the parentheses come for free. `\Next` and `\Last` refer to the next and the previous example without a label:

```
\ex.\label{ex:sleep} Ich habe geschlafen.  
\z.  
  
As \ref{ex:sleep} shows, ... The contrast in \Next\ is sharper:  
  
\ex. *Ich habe gestorben.  
\z.
```

renders as:

(5) Ich habe geschlafen.

As (5) shows, ...The contrast in (6) is sharper:

(6) *Ich habe gestorben.

This concludes the presentation of the essential syntax for the most elementary needs. The rest of this manual fills in the details: more nesting levels, more gloss tiers, arbitrary judgment marks, reference ranges, examples in footnotes, source attributions, and the layout parameters.

2 Origins, acknowledgments, and relation to other packages

linguex (Wolfgang Sternefeld). This package started from a set of patches I made to `linguex` over the years. As a consequence, the input language of `linguexx` *is* the input language of `linguex`, deliberately and almost exhaustively: `\ex.`, `\a.-\f.`, `\z.`, termination by blank line, custom labels in brackets, automatic roman numbering in footnotes, the `\Next/\Last` family, `\exg.` and its relatives, and the layout parameters down to their names. Existing `linguex` documents compile essentially unchanged; the default geometry is a little tighter than `linguex`'s, and the option `legacy` reproduces the original spacing exactly (section 8). What has been replaced is the machinery underneath. `linguex` tracks sub-example depth in a global counter that drifts out of step whenever an example is malformed, and the damage then propagates to the numbering of every later example; it recognizes judgments with a hand-written tokenizer that manipulates category codes; and it scans an example body up to the next `\par`, so that an example may not end a `beamer` frame. In `linguexx` the depth is held in the grouping structure itself, so drift is not merely unlikely but impossible; judgments are recognized by token inspection with no catcode trickery, and any mark whatever may be used (section 4); and the body is collected by a scanner that stops at a blank line, at `\z.`, or at a structural boundary such as `\end{frame}`. Two facilities of `linguex` were dropped as more trouble than they are worth: examples embedded inside sub-examples, and the index variants `\exi./\ai..`

cgloss4e (Hans-Peter Kolb and Craig Thiersch). The glossing commands `\gll`, `\glll` and `\glt` come from here, by way of `linguex`. The engine behind them is new, and the vertical space that `cgloss4e` inserted around a gloss is gone by construction.

expex (John Frampton). `expex` is the most capable glossing package in the field, and on that one axis `linguex` was simply outclassed: two or three tiers against an arbitrary number. `\gl ... \endgl` (section 5) closes that gap — any number of tiers, each with its own font command. It does not close every gap: `expex` retains finer control over the vertical layout of individual tiers, and its `keyval` interface exposes more parameters than the eight lengths of section 8. If the glossing is the hard part of your document, `expex` may still be the better instrument.

leipzig (Natalie Weber). This package contains a set of glossing conventions (the Leipzig glossing rules) that were lifted from Natalie Weber's package.

gb4e. Not an ancestor of the implementation, but its environment syntax is accepted: `\exe`, `\xlist` and the bracketed judgment form of `\ex` work as they do there (section 3.5). It is worth knowing that `gb4e` makes `_` and `^` active in text mode, which is a frequent source of clashes with other packages; `linguexx` changes no category codes.

Parenthesis-free references. The `\pref/\pNext` family follows a construction of Alan Munn’s.

Contributors This package has been coded and documented via generative AI (Claude), under the instruction and supervision of Gerhard Schaden.

3 Examples

3.1 Starting and ending an example

`\ex.` starts a numbered example. Three things end it:

1. a **blank line** (or an explicit `\par`) — the only thing that ended an example in `linguex`;
2. `\z.` (section 3.3), which ends it at the point where it stands;
3. a **structural boundary**: the end of the enclosing environment or group. An example may therefore run directly into `\end{<frame>}` or the closing brace of a group, with no blank line at all.

Environments *inside* an example do not end it: a `\tabular`, a `\tikzpicture` or a `\math display` is passed through untouched.

3.2 Sub-examples

`\a.` opens a level of sub-examples; `\b.` through `\f.` add further items at the level currently open. The letters printed come from a counter, not from the command name, so `\b.` prints “b.” only because it happens to be the second item — the commands are interchangeable and purely mnemonic. A second `\a.` opens a deeper level, numbered in roman:

```
\ex.
\ a. First.
\ b. Second.
\ a. First of the deeper level.
\ b. Second of the deeper level.
\ c. Back at the letter level? No -- see below.
```

renders as:

- (7)
- a. First.
 - b. Second.
 - i. First of the deeper level.
 - ii. Second of the deeper level.
 - iii. Back at the letter level? No – see below.

The last `\c.` comes out as item iii: the continuation commands stay at the level currently open. Two sub-levels are the maximum; a third `\a.` raises an error. To *leave* a level, use `\z..`

3.3 Returning to a higher level: `\z.`

`\z.` pops exactly one level, counting the surrounding prose as the outermost level. From the roman level it returns to the letters, so that a following `\b.` continues there; from the letter level — or in an example with no sub-examples open — it ends the example, and what follows is ordinary prose. Consecutive `\z.`'s therefore pop successively out of the example:

```
| \ex.  
| \a. First  
| \b. Second  
| \a. Second a  
| \b. Second b  
| \z.  
| \b. Third  
| \z.  
| Ordinary text again.
```

renders as:

- (8) a. First
 b. Second
 i. Second a
 ii. Second b
 c. Third

Ordinary text again.

The first `\z.` leaves the roman level, so `\b.` yields item c; the second leaves the example. Text on the same line as an example-ending `\z.` — like “Ordinary text again” above — is a *continuation*: it is set flush left under the example, without a paragraph indent. To start a genuinely new paragraph instead, indented as the class prescribes, leave a blank line after `\z.:`

```
| \ex. An example. \z.  
|  
| A new, ordinarily indented paragraph.
```

renders as:

- (9) An example.
- A new, ordinarily indented paragraph.

3.4 Custom labels

An optional argument replaces the number, without advancing the counter — useful for repeating an earlier example, or for primed variants:

```
| \ex.[(7$'$)] A repeated or modified example.
```

renders as:

- (7') A repeated or modified example.

3.5 An environment interface

Everything above can equally be written with conventional environments, whose syntax is that of `gb4e`. They are enabled by a package option: `\usepackage[gb4e]{linguexx}` loads *only* this syntax — the dot commands are then not defined, which keeps documents and error messages clean, and as a side benefit leaves the kernel accent commands `\b`, `\c`, `\d` untouched, so no `hyperref` workaround is needed. The default (equivalently, the option `lazy`) loads only the dot syntax of the preceding sections. The two may be combined, `[lazy,gb4e]`, for a document that deliberately mixes them — this manual does — and then the two forms drive the same engine, sharing one counter, one label system and all layout parameters. These options choose the input *syntax*; a further, orthogonal option `legacy` chooses the *geometry* of the original `linguex` and may be added to either (section 8). `exe` is a *batch*: every `\ex` inside it — written *without* the period — is a new top-level example. `xlist` embeds a sub-level, nestable once more for the romans; its items are made by `\ex` (or by plain `\item`). A judgment is given in the `gb4e` manner as an optional argument, `\ex[<judgment>]{<text>}`, where the judgment may be *any* mark, not only the auto-detected four — or typed directly after a plain `\ex`, as everywhere else:

```
\begin{exe}  
\ex\label{here} Here is one.  
\ex[*]{Here another is.}  
\ex Here are some with judgements.  
  \begin{xlist}  
    \ex[] {A grammatical sentence am I.}  
    \ex[??]{A dubious sentence is she.}  
  \end{xlist}  
\end{exe}
```

renders as:

- (10) Here is one.
- (11) *Here another is.
- (12) Here are some with judgements.
 - a. A grammatical sentence am I.
 - b. ?? A dubious sentence is she.

Two differences follow from the syntax itself: the environments end at their `\end` and nothing else, so blank lines inside them are ordinary paragraph breaks; and `\z.` does not end a batch, since an environment names its own end. An `xlist` may also sit inside a dot-command example, and `\a.` works inside `exe`. Use whichever form suits the document — or the editor.

Mixing the two syntaxes brings one rule with it, and it is the rule of the `xlist` environment rather than of the dot syntax: *inside a batch*, `\ex` continues at the level currently open. So once an `\a.` has opened a sub-level, a following `\ex` is the next *sub*-example, not the next top-level one. `\z.` is what closes the sub-level again — the one job it does have inside `exe`:

```

\begin{exe}
\ex One.
\a. A sub-example.
\z.
\ex Two, back at the top level.
\end{exe}

```

renders as:

- (13) One.
 a. A sub-example.
- (14) Two, back at the top level.

Without the `\z.`, “Two” would come out as sub-example b. Nothing warns about it, because that is what `\ex` means at the letter level: the same thing `\item` means inside an `xlist`. A `\z.` at the *top* level of a batch, where there is no sub-level left to close, is a package error naming `\end{exe}` — ending the batch is the environment’s own job.

3.6 Examples in footnotes

Inside a footnote, examples are numbered independently, in roman numerals, on their own counter; the running-text numbering is not disturbed, and references made inside the footnote resolve against the footnote’s series. Nothing special needs to be done: write `\ex.` as usual.¹

4 Grammaticality judgments

This section deals with the typesetting; marks are also given *spoken forms* for accessible PDFs, so that a screen reader announces a judgment by its meaning rather than by its glyph (section 9.1). That is one of the tagging features drawn together in section 9.

4.1 Automatic marks

The characters `*` and `?` and the commands `\#` and `\%`, in any combination, are recognized automatically when they open an example or sub-example. They are set in the margin, hanging to the left of the example text, so that judged and unjudged examples align:

```

\ex.
\a. Ich habe geschlafen.
\b. ??Ich habe geschlaft.
\c. *Ich bin geschlafen.
\d. \#Ich schlafte.

```

renders as:

-
- ¹Like so:
- (i) First footnote example.
- (ii) * Second one, with a judgment.

and (ii) refers to the second of these, not to a text example.

- (15) a. Ich habe geschlafen.
 b. ?? Ich habe geschlafen.
 c. * Ich bin geschlafen.
 d. # Ich schlafte.

The mark must come first in the item, *before* anything else including a `\label`. The conventional readings are the usual ones (* ungrammatical, ? degraded, # semantically or pragmatically deviant, % varying across speakers), but the package attaches no meaning to them: it only places them.

4.2 Space for judgments

A hanging mark takes no horizontal space, so it can never push the example text out of alignment — but it can, if long enough, run into the example *label* to its left. The room available is the clear space inside the label box plus the label gap, i.e.

$$\langle labelwidth \rangle - \text{width of the printed label} + \text{\Exlabelsep}.$$

At the main level the number box (`\Exlabelwidth`, 2.6em) leaves ample room. At the sub-levels the default widths (`\SubExlabelwidth` 1.6em, `\SubSubExlabelwidth` 1.9em) are chosen so that **two marks built from ? and *** (??, ?*, ...) fit clear of the letter. Note that `\#` and `\%` are nearly twice as wide as `?` in most fonts, so they count roughly double. If you use **three marks** (or wide marks) on sub-examples **as a routine matter**, widen the boxes:

```
\setlength{\SubExlabelwidth}{2.4em}
\setlength{\SubSubExlabelwidth}{2.5em}
```

which carries `?*#` comfortably. (An occasional long mark that grazes the letter is harmless; only alignment is guaranteed, and that unconditionally.)

4.3 Arbitrary marks: `\jdg`

For any other mark, `\jdg{<mark>}` hangs an arbitrary symbol into the margin, at any position, not only at the start of an example:

```
\ex. \jdg{\dag}A mark of one's own choosing.
\ex. \jdg{\$to\$}Or an arrow, or anything else.
```

renders as:

- (16) † A mark of one's own choosing.

- (17) → Or an arrow, or anything else.

Because the mark occupies no horizontal space, alignment is preserved whatever its width. To give a recurrent mark a name of its own, use `\DeclareJudgment{<command>}{<mark>}`:

```
\DeclareJudgment{\dubious}{\dag\dag}
\ex. \dubious A doubly daggered example.
```

renders as:

- (18) †† A doubly daggered example.

The distance between mark and text is the length `\JdgSep` (default 0.15em).

5 Interlinear glosses

5.1 Two and three tiers

`\gll` takes an object line and a gloss line, each ended by `\\`; `\glll` takes three lines. `\glt` introduces the free translation, which is set as an ordinary line, not aligned:

```
\ex. \gll Der Hund hat geschlafen \\
      the dog has slept           \\
\glt `The dog slept.'
```

renders as:

(19) Der Hund hat geschlafen
 the dog has slept
 ‘The dog slept.’

Words are separated by spaces, and *braced material counts as one word*. This is how one object-language word is glossed by several English words, or the reverse:

```
\ex. \gll Das {kleine Kind} schläft \\
      the {little child} sleeps      \\
\glt `The little child sleeps.'
```

renders as:

(20) Das kleine Kind schläft
 the little child sleeps
 ‘The little child sleeps.’

If the tiers are of unequal length, the missing cells are simply left empty — no error. A gloss too long for the line wraps between columns, never inside one.

5.2 Gloss abbreviations: `\lpzg`

Category labels in a gloss are written in small capitals by convention (SG, PST, NOM). `\lpzg{<label>}` is a convenience for exactly this: it sets its argument in small capitals, so `\lpzg{<sg>}` is a shorter, meaning-bearing spelling of `\textsc{<sg>}`. A whole compound label goes in one call, written the Leipzig way — a leading person number flush against the number, then further categories separated by periods:

```
\gll Der Hund bellte.\\
      the dog bark.\lpzg{3sg.pst}\\
\glt `The dog barked.'
```

renders as:

Der Hund bellte.
the dog bark.3SG.PST
‘The dog barked.’

`\lpzg{<3sg.pst>}` simply sets 3SG.PST. Its argument is a label you would otherwise have typed by hand; there is no fixed vocabulary as far as *typesetting* is concerned. What `\lpzg` adds over `\textsc` appears only in a tagged PDF, where it records the spoken expansion of the label (“third person singular past”) for a screen reader; that behaviour, and the list of abbreviations it knows, are described in section 9.2. `\lpzg` is self-contained and does not require, or interact with, the `leipzig` package.

5.3 The list of abbreviations used: `\lpzglst`

A paper that glosses its examples owes the reader a list of the abbreviations it uses. `\lpzglst` prints one, and prints exactly the abbreviations the document actually uses: every label passed to `\lpzg` is recorded, and a compound label is recorded piece by piece, so one `\lpzg{\textit{3sg.pst}}` contributes three entries — 3, SG and PST — and nothing else. Each entry is set as the abbreviation followed by its full form:

```
| \lpzglst
```

renders as:

3	third person
DEF	definite
M	masculine
NOM	nominative
PRS	present
PST	past
SG	singular

— which, here, is the list of everything this manual glosses with `\lpzg`, collected from its own examples. The record survives in the `.aux` file, so the list may stand where such a list belongs — in the front matter, before the examples it reports on. Like a table of contents it is then one run behind: `linguexx` asks for a rerun when the list is out of date. A list at the *end* of a document is complete on the first run.

Abbreviations that appear outside `\lpzg` — in running text, in a figure, in a table — are unknown to the recorder. Register them with `\lpzgadd{\textit{labels}}`, a comma list:

```
| \lpzgadd{erg,abs}
```

An abbreviation for which no expansion is known (a project-specific label neither in the built-in table of section 9.2 nor declared with `\SetLeipzig`) cannot be explained in the list, and is left out of it with a warning naming it. Either declare it, or ask for `unexplained=keep` to have it listed with an empty explanation.

Customisation

`\lpzglst` takes optional key–value settings for one list; `\lpzglstsetup{\textit{keys}}` sets them for all of them:

key	effect
style	list (the default): a two-column list of entries. inline : one paragraph, entries separated by sep .
sort	true (the default): alphabetically, the person numbers first. false : in order of first use.
include	used (the default) or all , which prints the whole built-in table — a ready-made reference list.
ignore	Comma list of labels to leave out.
add	Comma list of labels to include in this list only (unlike <code>\lpzgadd</code> , which registers a label for every list).
unexplained	omit (the default) or keep ; see above.
title	A heading for the list; none by default.
titlestyle	The one-argument command that sets the heading, by default <code>\lpzglsttitle</code> (<code>\section*</code> where that exists).
sep	Separator of the inline style, by default a semicolon.
itemsep	Vertical space between entries of the list style, <code>0pt</code> by default.
format	The entry itself, #1 the abbreviation and #2 its full form.

So a list headed like a section, with the abbreviations set in bold roman rather than the small capitals of `\lpzg`:

```
\lpzglst[title=Abbreviations,
      format={\item[\textbf{#1}] #2}]
```

`format` is the one-shot form of `\lpzglstentry`, which the **list** style calls with an `\item` and the **inline** style without one; redefine it with `\renewcommand` to change every list at once. The label column of the **list** style is as wide as the widest abbreviation in it.

5.4 Checking the abbreviations: `\lpzgcheck`

A mistyped abbreviation is invisible: `\lpzg{<pres>}` for `\lpzg{<prs>}` simply sets **PRES** in small capitals and says nothing. `\lpzgcheck` turns that into a warning at the end of the document, whether or not the document builds a list:

```
\lpzgcheck{unknown=true,      % on by default
            unused=true,      % off by default
            ignore={proj,adhoc}}
```

unknown reports every abbreviation used with no known expansion — not in the Leipzig table and not declared with `\SetLeipzig`. It is *on by default*, since such a key is nearly always a typo or a forgotten declaration. Abbreviations you mean to leave unexplained go in **ignore**.

unused reports the reverse: an abbreviation declared with `\SetLeipzig` and never used, which is what a stale declaration looks like. It is *off* by default, because keeping a standing set of declarations in a shared preamble and using only some of them in any one paper is a perfectly reasonable way to work. Only your own declarations are considered; the built-in table is not.

The two are independent of `\lpzglst`: it reports the keys it cannot explain in a list it builds, whereas these check the document. A key that a list has already reported is not reported twice.

5.5 Any number of tiers: `\gl ... \endgl`

Heavily annotated data — transliteration, segmentation, gloss, category line — needs more than three tiers. `\gl` takes any number of lines, each ended by `\`, up to `\endgl`:

```
\GlossTierFont{4}{\textit}%
\ex. \gl Der Hund schläft \
      der Hund schlaf-t \
      \lpzg{def.nom.sg.m} dog sleep-\lpzg{3sg} \
      D N V \ \endgl
\glit `The dog sleeps.'
```

renders as:

(21) Der Hund schläft
 der Hund schlaf-t
 DEF.NOM.SG.M dog sleep-3SG
 D *N* *V*
 ‘The dog sleeps.’

All the conventions of `\gll` carry over: spaces separate words, braces group them, unequal tiers get empty cells, long examples wrap between columns. `\gll` and `\glll` are in fact abbreviations of this one engine.

5.6 Fonts and spacing

Each tier is set with a one-argument command, declared by `\GlossTierFont{<n>}{<command>}`. Tiers 1–3 have traditional names of their own — `\eachwordone`, `\eachwordtwo`, `\eachwordthree` — which may be redefined instead; all default to `\textnormal`, so glosses inherit the surrounding font (under `beamer`, they come out sans-serif, as they should). To set the gloss line in small capitals throughout:

```
\renewcommand{\eachwordtwo}[1]{\textsc{#1}}
```

The space between columns is the macro `\GlossSep`, a glue specification (default `.5em` plus `.3em` minus `.1em`), changed with `\renewcommand`.

5.7 Aligning past brackets and judgment marks

When an object word opens with a bracket, a parenthesis or a judgment mark, the gloss word below it normally lines up with that mark, not with the word it introduces. Thus

```
\ex. \gll Ich bin [<ein Idiot>]\
      I am a idiot\
```

renders as:

(22) Ich bin [<ein Idiot>]
 I am a idiot

Here the gloss word **a** sits flush under the `[`, far to the left of **ein**. Switching on *phantom alignment* makes the gloss word skip the leading run of marks, so its first real glyph sits under the object word’s first real glyph (**a** under **ein**):

```
{\GlossPhantomAlign
\ex. \gll Ich bin [<ein Idiot>]\\
I am a idiot\\
}
```

renders as:

(23) Ich bin [<ein Idiot>]
I am a idiot

The padding is a `\phantom` of the leading marks *set in the object-line font*, so alignment holds even when the gloss tier is set at a different size (a `\footnotesize` gloss still lines up under the full-size bracket). It ships no ink and no marked content, so tagging is unaffected. Any number of leading marks are skipped together; the marks that count are, by default, the judgment marks `*?#%` and the openers `([<`. Change the set with `\GlossPhantomChars{<chars>}`, e.g. `\GlossPhantomChars{[(}` to react to square and round brackets only.

The feature is *off by default* — only because turning it on shifts the horizontal position of glosses under bracketed material, and existing documents should keep the layout they were written for. For new documents it is *recommended*: it is what makes an interlinear gloss line up the way a reader expects, and it costs nothing under tagging. Enable it document-wide with the package option `phantomalign` or with `\GlossPhantomAlign`; `\GlossPhantomAlignOff` turns it back off, so it can be scoped to a single example inside a group. It applies to the space-separated words of `\gll/\glll/\gl`; alternatives stacked with `\altg` are not covered.

For anything the automatic scan cannot see — a bracket wrapped in a macro (the scanner reads the control word, not the `[]`), or a target you want to choose by hand — there is a manual escape hatch: `\GlossPhantom{<material>}` typesets an invisible box the width of `<material>`, *set in the object-line font*. Put it at the front of a gloss word to push that word past `<material>`:

```
\ex. \gll Das ist \textbf{[<ein Reh\\
This is \GlossPhantom{\textbf{[<}a deer\\
```

renders as:

(24) Das ist [ein Reh
This is a deer

Give it whatever the object word opens with (here `\textbf{[<}`, so the phantom is a *bold* bracket and the widths match). Unlike a bare `\phantom{<material>}`, which would be set in the gloss font, `\GlossPhantom` borrows the object font, so it aligns at any gloss-tier size. One caveat: because it reproduces the width of the leading material *alone*, alignment is exact only when that material does not kern with the letter that follows it in the object word. Brackets, parentheses and judgment marks do not kern with letters, so in practice this never bites; a leading slash or the like might land a fraction off.

5.8 The language of a tier (tagged PDFs)

The object line of a gloss is usually in a different language from the document. In a tagged PDF that difference can be recorded so that a screen reader pronounces each tier with

its own phonetics rather than reading, say, German words as though they were English. Declare the language of a tier with `\GlossTierLang{<tier>}{<code>}`, where *<code>* is a language tag such as `de` or `fr`:

```
| \GlossTierLang{1}{de}
| \ex.
| \gll Der Hund bellte.\\ the dog barked.\\
| \glt `The dog barked.'
```

Each word of tier 1 is then wrapped, in the structure tree, in a span carrying that language; the other tiers are unaffected.

`\GlossTierLang` obeys the usual scope rule. Given in the preamble or in the document body between examples, it sets a *document-wide default* for that tier. Given *inside* an example, before the gloss, it overrides that tier for that one example only and reverts afterwards, so a document that glosses several object languages needs only a local `\GlossTierLang` in the odd example that departs from the default:

```
| \GlossTierLang{1}{de}           % default: object line is German
| ...
| \ex.
| \GlossTierLang{1}{fr}           % this example only
| \gll Le chien aboya.\\ the dog bark.\lpzg{pst}\\
| \glt `The dog barked.'
```

As with the rest of the tagging support this is inert unless tagging is active (section 9): it changes nothing in the printed gloss and nothing in an untagged document. A tier with no declared language behaves exactly as before.

The free translation is a tier of its own kind, and usually in a third language: neither the object language nor, in a paper written in one language and glossing into another, the language of the document. `\GlossTransLang{<code>}` marks it, and under tagging the translation is wrapped in a span carrying `/Lang`:

```
| \GlossTierLang{1}{de}           % object line is German
| \GlossTransLang{en}             % free translations are English
```

This is worth setting explicitly even in a document that already declares its language to `\DocumentMetadata: babel's \foreignlanguage` does *not* reach the structure tree, so without `\GlossTransLang` a translation in another language carries no `/Lang` of its own and is read out with the document's phonetics.

The translation can also be styled, with the declaration `\GlossTransStyle` — many venues want it italic, or a size smaller:

```
| \renewcommand\GlossTransStyle{\itshape}
```

It applies only to the translation, not to the gloss tiers above it, and reverts at the end of the example. Both hooks are empty by default, so a document that sets neither is unaffected in its output and in its tagging alike.

5.9 Abbreviations: `\exg.` and `\ag.`

`\exg.` abbreviates `\ex.` `\gll`, and `\ag.` through `\fg.` abbreviate `\a.` `\gll` and so on, for glossed sub-examples. Judgments still work in front of the gloss:

```
\exg. *Das Beispiel funktionieren \\
      the example work.INF          \\
\glt (intended: `The example works.')
```

renders as:

(25) *Das Beispiel funktionieren
 the example work.INF
 (intended: ‘The example works.’)

6 Cross-references

6.1 Labels

Examples and sub-examples are ordinary L^AT_EX counters: `\label` them and `\ref` them. The reference prints with parentheses, and a sub-example reference includes its letter:

```
\ex.\label{ex:main}
\ a.\label{ex:a} Erste.
\ b.\label{ex:b} Zweite.

See \ref{ex:main}, and in particular \ref{ex:b}.
```

renders as:

(26) a. Erste.
 b. Zweite.

See (26), and in particular (26b).

6.2 References without parentheses

In running text one often wants “example 3 b” rather than “example (3 b)”. Every reference command has a p-prefixed twin that omits the parentheses: `\pref`, `\pNext`, `\pLast`, `\pNNext`, `\pLLast`, `\pTextNext`.

```
| \ref{ex:b} with parentheses, \pref{ex:b} without.
```

renders as:

(26b) with parentheses, 26b without.

6.3 Relative references

To refer to a neighbouring example without labelling it:

Command	Refers to
<code>\Next</code>	the next example
<code>\NNext</code>	the one after that
<code>\Last</code>	the previous example
<code>\LLast</code>	the one before that
<code>\TextNext</code>	the next example in the running text (for use inside footnotes)

Inside a footnote these refer to the footnote’s own series, which is nearly always what is meant; `\TextNext` escapes to the main series.

These commands are part of the `linguex` compatibility surface and are kept for it, but they are worth using sparingly. What they resolve to depends on where they stand, so inserting an example between a reference and its target silently changes what the reference means — and nothing in the document records that anything was meant at all. In a document you expect to revise, prefer `\label` with `\ref` or, better, `cleveref`’s `\cref` (below): a label survives being moved, and a label that has lost its target is an error rather than a wrong number.

6.4 Ranges

`\refrange{<first>}{<last>}` sets a compact range over sub-examples, printing “(3a–c)” rather than “(3a)–(3c)”. The sub-examples must be labelled with `\sublabel` instead of `\label` (this records the label in the `.aux` file; `\sublabel` also does everything `\label` does, so `\ref` keeps working). It works at either sub-level, recording the letter at the letter level and the numeral at the roman one, so a range over sub-sub-examples closes with a numeral: “(3b–i–iii)”. On a main-level example it records nothing, and the range closes with a full reference instead:

```
\ex.
\ a.\sublabel{r:a} Erste.
\ b. Zweite.
\ c.\sublabel{r:c} Dritte.

The examples in \refrange{r:a}{r:c} all show ...
```

renders as:

- (27) a. Erste.
 b. Zweite.
 c. Dritte.

The examples in (27a–c) all show ...

cleveref. If `cleveref` is loaded, `\cref` works on examples and sub-examples. It prints the number alone — “(1)”, “(1a)” — rather than putting a word in front of it, since that is how examples are referred to in running prose; what it adds over `\ref` is its handling of several references at once, `\cref{a,b}` giving “(1) and (2)”. Should you want a word after all, say so in the preamble and it is left alone:

```
\crefname{ExNo}{example}{examples}
```

The counters are `ExNo`, `SubExNo`, `SubSubExNo` and `FnExNo`.

`\crefrange` works too, but it is not a replacement for `\refrange`: over the first and last sub-example of an example it prints “(3a) to (3c)”, where `\refrange` prints “(3a–c)”. `cleveref` has no way to compress a range whose members share a prefix, which is the compact form a linguistics text wants, so `\refrange` and `\sublabel` are still the way to get it — and a label may carry both, since `\sublabel` does everything `\label` does.

`\prefrange` is the parenthesis-free variant; `\Refrange` sets a range over two whole examples, as (10)–(12). The range dash is the macro `\rangedash` (default an en-dash).

The dash between number and letter is `\firstrefdash` — empty by default, so “(3a)”, but `-` under `legacy` — and between letter and roman numeral `\secondrefdash` (`-` in both modes); see section 8.

7 Further apparatus

`linguexx` contains three constructs that were not part of its ancestor `linguex`: `\altn`, `\altg`, and `\exsource`.

7.1 Stacked alternatives: `\altn`

A set of interchangeable constituents is conventionally stacked inside a curly brace. `\altn` takes any number of brace groups and stacks them:

```
\ex. \altn{\sout{This girl in the red coat}}{She}{Mary} will put a
picture of \altn{Bill}{him} on your desk.
```

renders as:

$$(28) \quad \left\{ \begin{array}{c} \text{This girl in the red coat} \\ \text{She} \\ \text{Mary} \end{array} \right\} \text{ will put a picture of } \left\{ \begin{array}{c} \text{Bill} \\ \text{him} \end{array} \right\} \text{ on your desk.}$$

The alternatives are collected as long as brace groups follow one another immediately; the first non-brace token ends the list. Strike-through (`\sout`) is available for an excluded alternative if your document loads `ulem` — the package does not load it for you. An optional argument sets the alignment: `[c]` (default), `[l]`, `[r]`.

A guaranteed-available fallback. `\altn` was chosen because nothing in a current $\text{\TeX}$ Live distribution defines it (unlike the shorter `\alt`, which `beamer` and several other packages already claim). If some other package you load ever does define `\altn` first, `linguexx` detects this, leaves that command alone, notes it in the log, and falls back: `\write \lxAltn` instead. Otherwise `\altn` and `\lxAltn` are the same command.

For a light word-level alternation inside a glossed line a slash (`gehe/laufe`) is often enough; when each alternative needs a gloss of its own, use `\altg` (section 7.2).

`\altn` is set in text mode: the alternatives are the rows of a `tabular`, braced on both sides with braces drawn with `TikZ`, so no mathematics is involved (the package needs `graphicx` and `tikz`, not `amsmath`). The braces are tunable through five macros, redefined with `\renewcommand`. Two govern the drawn shape of the bracket itself: `\AltBraceWidth` (default `0.9ex`) is the horizontal box reserved for one brace glyph — the vertical spine of the `{/}` is drawn close to the edge of this box that faces the stack, so widening `\AltBraceWidth` leaves more room on the far side for the curl to bulge into; and `\AltBraceAmplitude` (`4pt`) is passed straight through as `TikZ`’s own `brace` decoration `amplitude` key — how far the small pointed tip at mid-height projects sideways from that spine, so a larger value draws a more pronounced curl and a smaller one flattens the bracket toward a plain vertical stroke. The other three govern placement, not shape: `\AltBraceSep` (`0.15em`, the gap between brace and stack), `\AltBraceOuterSep` (`0.7em`, the gap between the brace and whatever precedes or follows it — wider than `\AltBraceSep` because the drawn curl bulges outward past its own box only on that side), and `\AltBraceRaise` (`0pt`, a vertical nudge). `\altg` (section 7.2) draws its braces with the same `\AltBraceWidth` and `\AltBraceAmplitude`, so changing either one here reshapes both commands’ brackets together.

Because the alternatives are ordinary text rather than a formula, they are read sensibly in a tagged PDF. Under active tagging `\altn` wraps the stack in a span carrying a spoken form of the list — “This girl in the red coat, She, or Mary” — built from the alternatives themselves, with any formatting (such as the `\sout` above) stripped for speech. The printed output is unchanged, and an untagged document is unaffected (section 9).

7.2 Glossed alternatives: `\altg`

Where `\altn` stacks bare constituents, `\altg` stacks a paradigm in which each alternative carries its own gloss. Inside an interlinear gloss it is written *twice* — once in the object line with the object words, once in the gloss line with their glosses:

```
\exg. Die \altg{Frau}{Socke}{Maus}{Tonne} ist hier.\
      The.\lpzg{sg} \altg{woman.\lpzg{sg}}{sock.\lpzg{sg}}{%
      {mouse.\lpzg{sg}}{ton.\lpzg{sg}} is.\lpzg{prs} here.\
```

renders as:

(29)	Die	{	Frau	woman.SG	}	ist	hier.
	The.SG	{	Socke	sock.SG	}	is.	PRS here.
		{	Maus	mouse.SG	}		
		{	Tonne	ton.SG	}		

The two calls occupy the two tiers of one gloss column and assemble a single paradigm: object stack on the left, gloss stack to its right (offset by `\AltgColSep`), braced on *both* sides. The block is centred on the midline between the object tier and the gloss tier: with four alternatives, rows two and three ride the object and gloss lines and the outer rows protrude symmetrically, with the surrounding lines keeping clear. The example number stays on the object baseline, where `\exg.` puts it.

Four rules of use. The two calls must list the same number of alternatives — the package stops with an error otherwise. They must also fall in the *same* gloss column, one in the object tier and one in the gloss tier: a column carrying a stack in only one of its tiers cannot be paired up, and is likewise an error. (This is why a three-tier `\glll` cannot take a stack in its third tier either. Alternatives spread over three tiers are not supported.) No spaces may separate the brace groups: a space both ends the collection and splits the gloss into separate columns; break long calls across source lines with `%`, as above. And the alternatives are paired by position — the *n*-th gloss glosses the *n*-th object word.

Outside a gloss, a single `\altg` sets one both-braced stack on the current baseline — `\altn` with a closing brace, in effect.

Because each stack already competes for both tiers, `\lpzg` inside it prints as plain small caps, without the abbreviation span of section 9; the expansions are not lost — they reappear in the spoken forms. Under active tagging each call is wrapped in a span carrying a spoken `/Alt` of its own list — “Frau, Socke, Maus, or Tonne” for the object call, “woman.singular, sock.singular, mouse.singular, or ton.singular” for the gloss call — with simple `\lpzg` keys expanded from the Leipzig table; compound keys (`3sg.pst`) and unknown keys are spoken as printed. Like `\altn`, none of this is mathematics, so the alternatives remain ordinary tagged text.

Two settings of its own, redefined with `\renewcommand`: `\AltgColSep` (default 1.2em), the gap between the object and gloss stacks, and `\AltgTransFont` (default `\normalfont`), the font of the gloss stack. The braces are drawn with the `\altn` tunables (`\AltBraceWidth`, `\AltBraceAmplitude`, `\AltBraceSep`, `\AltBraceOuterSep`). If some other package owns `\altg`, the package leaves it alone and only `\lxAltg` is available, as with `\altn`/`\lxAltn`.

7.3 Source attributions: `\exsource`

`\exsource` sets an attribution flush right at the end of an example, dropping it onto a line of its own if it does not fit:

```
\ex. This girl in the red coat will put a picture of Bill on your desk.
\exsource{(Perlmutter 1971: 76)}
```

renders as:

(30) This girl in the red coat will put a picture of Bill on your desk. (Perlmutter 1971: 76)

It works in every part of a multipart example, and after a gloss. The font is the hook `\ExSourceFont` (default `\normalfont\footnotesize`); the argument may contain a `\cite`.

8 Example and glossing layout

The geometry is uniform across levels: each level reserves a label box of fixed width, followed by the shared gap `\Exlabelsep`; the text margin of a level is its label width plus `\Exlabelsep`, measured from the margin of the level above (figure 1). Judgments hang to the left of the text margin and take no space.

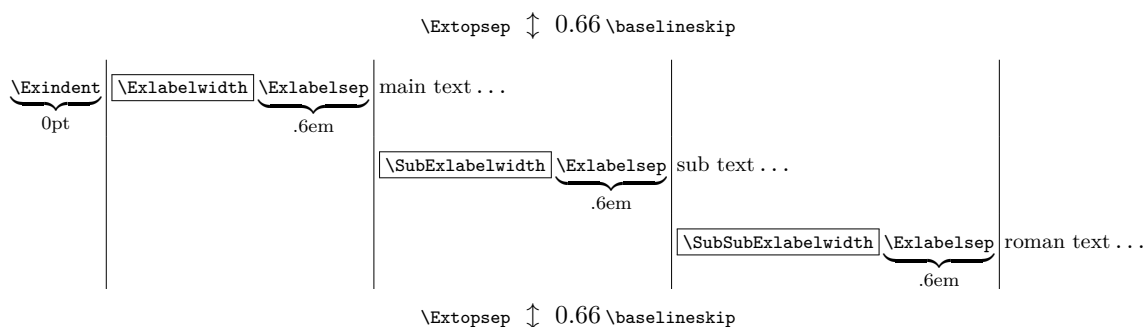


Figure 1: The default box model. On each line, reading left to right: an optional `\Exindent` (0pt), the label box, then the shared gap `\Exlabelsep`; the text of a level begins one label box plus one `\Exlabelsep` to the right of the level above, so the boxes cascade down the page. Each box is named for the length that fixes its width (`\Exlabelwidth` 2.6em, `\SubExlabelwidth` 1.6em, `\SubSubExlabelwidth` 1.6em) and holds the number, the letter and the roman label respectively; the vertical rules mark the three text margins. `\Extopsep` is the space left above and below. Under `legacy` the sub-levels are driven instead by two leftmargins, `\SubExleftmargin` and `\SubSubExleftmargin`, as in `linguex`.

Every parameter is an ordinary length, set with `\setlength` anywhere in the document:

Length	Default	Role
<code>\Extopsep</code>	<code>.66\baselineskip</code>	Vertical space before and after an example.
<code>\Exredux</code>	<code>-.66\baselineskip</code>	Correction inserted between <i>consecutive</i> examples, so the gap is not doubled. Keep it at <code>-\Extopsep</code> for an even rhythm.
<code>\Exlabelwidth</code>	<code>2.6em</code>	Width of the number box — room for “(199)”. Widen it for four-digit numbering.
<code>\Exlabelsep</code>	<code>.6em</code>	Gap between any label box and its text.
<code>\SubExlabelwidth</code>	<code>1.6em</code>	Width of the letter box (“a.”). Deliberately wider than the letter: the surplus is where a hanging judgment goes (section 4.2).
<code>\SubSubExlabelwidth</code>	<code>1.6em</code>	Width of the roman box, sized for the widest label up to “vi.” (“iii.”), which is the same width as the widest letter; equal to <code>\SubExlabelwidth</code> , so the two sub-levels share one box width.
<code>\JdgSep</code>	<code>0.15em</code>	Gap between a hanging judgment and the text.

A few further knobs are macros, not lengths, and are changed with `\renewcommand`: `\GlossSep` (glue between gloss columns) and the three reference dashes. `\rangedash` (default `--`) separates the two ends of a range; `\firstrefdash` sits between an example number and a sub-example letter in a reference such as “(3a)”, and `\secondrefdash` between the letter and a roman numeral in “(3a-i)”. Their defaults differ by mode: in the default scheme `\firstrefdash` is *empty* (giving “(3a)”) and `\secondrefdash` is `-`; under **legacy** both are `-`, so the same reference prints “(3-a)” and “(3-a-i)”, as *linguex* does. The sub-label delimiters `\SubExLBr`/`\SubExRBr` and `\SubSubExLBr`/`\SubSubExRBr` bracket the letter and roman labels themselves; they are `{}/\{.` in both modes for the letter (“a.”), while the roman is bare “i.” by default but parenthesised “(i)” under **legacy**.

The counters are `ExNo`, `SubExNo`, `SubSubExNo` and `FnExNo` (footnotes). To change the style of the numbering, redefine `\theExNo` and its relatives in the usual L^AT_EX way.

The default scheme and legacy. The values in the table are the *default* scheme. The option **legacy** selects instead the exact geometry and conventions of *linguex*: the number box is sized per example from the width of its digits rather than fixed at `\Exlabelwidth`; the sub-levels are positioned by two leftmargins, `\SubExleftmargin` (2em) and `\SubSubExleftmargin` (2.4em), rather than by a label width plus `\Exlabelsep`; roman sub-sub-labels print as “(i)”; `\firstrefdash` is `-`; and, in a book-class document, `ExNo` resets at each `\chapter`. **legacy** is orthogonal to the choice of input syntax: combine it freely, as in `[legacy,gb4e]`. Both parameter sets exist in either mode — `\SubExleftmargin` always equals `\SubExlabelwidth + \Exlabelsep` — so a document can be nudged from one scheme toward the other one length at a time.

`\resetExdefaults` restores every layout parameter to the values of the mode in force;

it is what the package runs for you as it loads. Because the defaults are applied at load time, not `\AtBeginDocument` as in `linguex`, a `\setlength` in your preamble simply wins — there is no need to hook it into `\resetExdefaults` to keep it from being overwritten. The two font-relative lengths, `\Extopsep` and `\Exredux`, are the exception: they are finalised at `\begin{document}` so that they follow the body font, unless you have already given either an explicit value.

9 PDF tagging and accessibility

`linguexx` is aware of the $\text{\LaTeX}$ project’s PDF tagging code. If the document begins with a `\DocumentMetadata` line that turns tagging on, for example, on a current $\text{\TeX}$ distribution,

```
| \DocumentMetadata{lang=en,tagging=on}
```

then examples are written into the PDF’s structure tree as genuine, accessible objects rather than as loose ink. Without such a line nothing below applies: the printed output is identical and the package behaves exactly as in the rest of this manual.

The exact spelling of that line may depend on your version of $\text{\LaTeX}$. The `tagging=on` key is the recommended form from the $\text{\LaTeX}$ release of November 2025 onward. On distributions between roughly 2023 and 2025 the tagging code lives behind the experimental `testphase` key instead; the portable choice there, which also works on current $\text{\TeX}$, is

```
| \DocumentMetadata{lang=en,testphase={phase-III}}
```

Both enable the same comprehensive tagging as far as `linguexx` is concerned.²

The structure tree of all structures provided by `linguexx` is valid, including the case of an example inside a footnote. Each example is a list item with its number as the label and its content as the body; the letter and roman sub-levels are nested lists inside it, so assistive technology can walk into and out of the hierarchy. Those lists are marked as *ordered*, matching their numbering. Grammaticality marks are given a spoken form, so a screen reader announces “ungrammatical” rather than “asterisk” (section 9.1). In an interlinear gloss each column — an object word together with its aligned gloss(es) — is grouped as one structure element, so the gloss is read and navigated word bundle by word bundle, in the object-then-gloss order, rather than as one undifferentiated run of text. The object language of a tier can be recorded (section 5.8), so it is pronounced with its own phonetics; and category abbreviations written with `\lpzg` carry their expansion (section 9.2), so a reader hears “past” rather than “pee-ess-tee”.³ Stacked alternatives (`\altn`) are set in text mode rather than as a formula, and carry a spoken form of the list (section 7), so they are read as “A, B, or C” rather than as a run of braces and glyphs.

A document built with the accessibility preamble above validates as PDF/UA-2: veraPDF reports it conformant, with every rule and check passing and no exceptions. Conformance is a document-level property, not only a matter of the examples — it also needs a document title and a few other pieces of metadata — so the package ships a short checklist (PDFUA-CHECKLIST) giving the preamble that supplies them. What a validator cannot certify is that the result is *pleasant* to listen to; that still wants testing with a real screen reader.

²Avoid the older habit of naming individual modules (`testphase={tagpdf,...}`): those names have been reorganised and can now load only part of the machinery.

³This also means that tagging is only useful for English at the current time.

A caution on stability. Tagging in L^AT_EX requires a recent L^AT_EX. `linguexx`’s support is written to degrade quietly — if the tagging machinery is absent or turned off, the relevant code does nothing — but it should be regarded as provisional and may need a touch-up as the tagging project matures. The `legacy` option is orthogonal to all of this; tagging support targets the default mode.

9.1 Spoken forms for tagged PDFs

A judgment mark is a symbol whose meaning a screen reader cannot guess: read literally, `*` is “asterisk”. When the document is compiled with the PDF tagging code active (a `\DocumentMetadata` line enabling the tagging `testphase`), `linguexx` wraps each mark in a structure element carrying an *alternate text* so it is announced by its meaning instead. The built-in marks have the following defaults:

mark	spoken as
<code>*</code>	ungrammatical
<code>?</code>	questionable
<code>??</code>	highly questionable
<code>?*</code>	extremely degraded
<code>#</code>	infelicitous
<code>%</code>	grammatical for some speakers

This affects only the tagged PDF’s accessibility layer; the printed output is unchanged, and nothing happens on an engine or in a document without active tagging. To set the spoken form of a mark, give `\DeclareJudgment` its optional `spoken` key — which both names the mark and registers its spoken form, so a leading scanned mark is announced the same way —

```
| \DeclareJudgment[spoken={marginal, some speakers}]{\pcent}{\%}
```

or, for a mark you do not need to name, `\SetJudgmentSpoken {<mark>}{<phrase>}`. An explicit `\jdg` may also carry a one-off spoken form as an optional argument:

```
| \jdg[contradictory]{*}
```

9.2 Gloss abbreviations and their expansions

Under active tagging, `\lpzg{<label>}` does more than set small capitals (section 5.2): it records the label’s *expansion* as the PDF “expansion text” (the /E entry of an abbreviation), so a screen reader announces “singular” while the page still shows SG and copy-and-paste still yields `sg`. This is the correct mechanism for an abbreviation — distinct from an *alternate text*, and unlike `/ActualText` it leaves the copied text alone.

A compound label is parsed for the expansion: it is split on periods, a leading 1, 2 or 3 is taken as the person, and each piece is expanded and joined into one phrase, so `\lpzg{<3sg.pst>}` reads “third person singular past”. A piece not in the table below passes through verbatim; a label with nothing recognisable in it gets no expansion, and no warning, so project-specific labels are safe. Extend or override the table with `\SetLeipzig{<label>}{<expansion>}`:

```
| \SetLeipzig{obv}{obviative}
```

The built-in table is the standard Leipzig Glossing Rules list; the meaning column is what a screen reader speaks.

label	meaning	label	meaning
1	first person	INDF	indefinite
2	second person	INF	infinitive
3	third person	INS	instrumental
A	agent	INTR	intransitive
ABL	ablative	IPFV	imperfective
ABS	absolutive	IRR	irrealis
ACC	accusative	LOC	locative
ADJ	adjective	M	masculine
ADV	adverbial	N	neuter
AGR	agreement	NEG	negative
ALL	allative	NMLZ	nominalizer
ANTIP	antipassive	NOM	nominative
APPL	applicative	OBJ	object
ART	article	OBL	oblique
AUX	auxiliary	P	patient
BEN	benefactive	PASS	passive
CAUS	causative	PFV	perfective
CLF	classifier	PL	plural
COM	comitative	POSS	possessive
COMP	complementizer	PRED	predicative
COMPL	completive	PRF	perfect
COND	conditional	PROG	progressive
COP	copula	PROH	prohibitive
CVB	converb	PROX	proximal
DAT	dative	PRS	present
DECL	declarative	PST	past
DEF	definite	PTCP	participle
DEM	demonstrative	PURP	purposive
DET	determiner	Q	question particle
DIST	distal	QUOT	quotative
DISTR	distributive	RECP	reciprocal
DU	dual	REFL	reflexive
DUR	durative	REL	relative
ERG	ergative	RES	resultative
EXCL	exclusive	S	argument of intransitive verb
F	feminine	SBJ	subject
FOC	focus	SBJV	subjunctive
FUT	future	SG	singular
GEN	genitive	TOP	topic
IMP	imperative	TR	transitive
INCL	inclusive	VOC	vocative
IND	indicative		

9.3 Transliteration and the engine you compile with

One trap is worth knowing about, because nothing warns you and the page looks perfect. Under pdfL^AT_EX, characters whose diacritic sits *below* the letter are not single glyphs: T_EX composes them by placing a period-like glyph under the base letter. The result prints

correctly, but the PDF’s text layer records the two pieces, so what copy-paste and a screen reader receive is not what you wrote:

you write	pdfL ^A T _E X extracts	XeL ^A T _E X/LuaL ^A T _E X extract
krṣṇaḥ (Indic)	kr.s.n.ah.	krṣṇaḥ
ḍāṇṭaḥ (Indic)	d . ān.t.ah.	ḍāṇṭaḥ
ģimeņu (Latvian)	gimen ,u	ģimeņu

This manual is itself built with LuaL^AT_EX, for exactly this reason: it contains those characters in the table above, and under pdfL^AT_EX it would exhibit the defect it is describing — copying the first column out of the PDF would give you the second. Built as it is, the first and third columns copy out as written.

This affects dot-below (Indic and Semitic transliteration: ḍ ṇ ṭ ṣ ḥ ṛ ṁ) and comma-below (Latvian ģ ķ ļ ņ, and the same command produces Romanian ș ț). Two things make it easy to miss. Tagging does *not* repair it — the mapping is supplied per glyph, and here there are two glyphs where you meant one — so it is present in a fully tagged ua-2 document as much as in a plain one. And veraPDF *passes* such a file on all three profiles: the structure tree is valid, so the only authoritative check for PDF/UA gives it a clean bill while the text a screen reader reads is wrong.

If your data uses those scripts, compile with XeL^AT_EX or LuaL^AT_EX, where the characters stay single glyphs and extract correctly. For those languages this is an accessibility requirement, not a preference. Accents *above* the letter — the whole European Latin repertoire, ä é ċ ö ž ø þ and the rest — are unaffected and extract correctly under all three engines.

10 Notes and limitations

- **Two sub-levels.** A third \a. is an error. Deeper hierarchies are almost always better expressed as separate examples.
- **The period is part of the command.** \ex., \a., \z. — not \ex, \a, \z. This is what allows the syntax to be so light.
- **\z. at brace depth 0.** \z. is recognized in the body of an example, not inside a brace group within it.
- **\verb in an example body.** It cannot work in the dot syntax. A dot-syntax body is *collected* before it is typeset — that is exactly what lets a blank line end it and \z. pop a level — and by then its tokens have long since been read, with the catcodes in force where the \ex. stood. \verb has to change those catcodes *while* reading, so it finds nothing left to protect and fails, usually as “Missing \$ inserted”. The same goes for verbatim, listings and fancyvrb’s environments, and for anything else that reads its own argument specially. This is inherent to a blank-line-delimited body rather than a defect to be fixed someday — linguex grabs its bodies at \par and has it too. The environment syntax does *not* collect, so there \verb works normally, including inside an \a. or an xlist written within the batch:

```
\begin{exe}
\ex A body with \verb|x_y| in it.
\end{exe}
```

The one exception is the braced judgment form `\ex[⟨judgment⟩]{⟨text⟩}`, whose body is a macro argument and is therefore read before it is used, exactly like a collected one. So: for verbatim material in an example, use `exe` with an unbraced `\ex`.

- **Examples in a minipage footnote.** They are numbered on the main series, not on the footnote series: `\footnote` inside a `minipage` is a different command from the ordinary one, and only the ordinary one is hooked. Whether such examples ought to share the footnote series, keep the main one, or get a series per minipage is a question `linguex` never answered either, so nothing is claimed here and nothing is patched; if the numbering matters in your document, put the examples outside the minipage.
- **ulem is not loaded.** For struck-through alternatives (`\sout` inside `\altn`), load `ulem` yourself; the usual choice is `\usepackage[normalem]{ulem}`, which keeps `\emph` italic.
- **The accents `\b`, `\c`, `\d` keep working.** These are accent commands as well as sub-example letters, and both meanings are available everywhere, including in the same example: each letter dispatches on what follows it, so a period gives the sub-example command and anything else the accent. `\c{⟨c⟩}` and a literal “`c`” — which `inputenc` turns into exactly that pair of tokens under `pdfLATEX` — both work in running text, in a section title and inside an example body. The same holds under `hyperref`, in either load order.
- **Shorthand names another package owns.** The glossed shorthands `\ag.`–`\fg.` are claimed at `\begin{document}` and only if the name is still free. `babel`’s French option defines `\fg`, the closing guillemet of `\og ... \fg`, so in a French document `\fg.` is simply not available and the guillemet keeps its meaning; write `\f.` followed by `\gll` instead. The same applies to any name you have defined yourself — `\eg` is a common one — and the fact is recorded in the `.log`.
- **Footnote examples.** End an example inside a footnote with a blank line or `\z.` rather than letting it run into the footnote’s own end.
- **Glosses and alternatives do not combine.** See section 7.
- **Where `expex` still does more.** Tier *count* and tier *fonts* are unrestricted here, but the vertical layout of a gloss is not individually adjustable per tier, and there is no `keyval` interface. See section 2.

11 Command reference

Command	Effect
<i>Examples</i>	
<code>\ex.</code>	Start a numbered example; a blank line ends it. <code>\ex.[<i>label</i>]</code> uses a custom label without advancing the counter.
<code>\a.</code>	Open a level of sub-examples (letters, then roman).
<code>\b. -- \f.</code>	Next item at the level currently open.
<code>\z.</code>	Leave one level; from the letter level or an example without sub-examples, end it.
<code>exe, xlist</code>	gb4e-compatible batch and sub-level environments (package option <code>gb4e</code>); items via <code>\ex</code> , <code>\ex[<i>judgment</i>]{<i>text</i>}</code> , or plain <code>\item</code> .
<code>\exg.</code>	<code>\ex.</code> followed by <code>\gll</code> .
<code>\ag. -- \fg.</code>	<code>\a.–\f.</code> followed by <code>\gll</code> .
<i>Judgments</i>	
<code>\jdg{<i>mark</i>}</code>	Hang an arbitrary mark in the margin.
<code>\DeclareJudgment{<i>cmd</i>}{<i>mark</i>}</code>	Give a mark a name.
<i>Glosses</i>	
<code>\gll, \glll</code>	Two-, three-tier gloss; lines end in <code>\.</code>
<code>\gl ... \endgl</code>	Gloss with any number of tiers.
<code>\glt</code>	Free-translation line. (<code>\gln</code> is a synonym.)
<code>\GlossTierFont{<i>n</i>}{<i>cmd</i>}</code>	Font command for tier <i>n</i> .
<code>\GlossTierLang{<i>n</i>}{<i>code</i>}</code>	Language of tier <i>n</i> for tagged PDFs (section 5.8).
<code>\GlossTransStyle</code>	Declaration applied to the free translation after <code>\glt</code> ; empty by default.
<code>\GlossTransLang{<i>code</i>}</code>	Language of the free translation for tagged PDFs (section 5.8).
<code>\lpzgj{<i>abbr</i>}</code>	Gloss abbreviation in small caps, with spoken expansion when tagged (section 5.2).
<code>\SetLeipzig{<i>abbr</i>}{<i>expansion</i>}</code>	Add or override a Leipzig abbreviation expansion.
<code>\lpzglst[<i>keys</i>]</code>	List of the abbreviations the document uses, each with its full form (section 5.3). Keys: <code>style</code> , <code>sort</code> , <code>include</code> , <code>ignore</code> , <code>add</code> , <code>unexplained</code> , <code>title</code> , <code>titlestyle</code> , <code>sep</code> , <code>itemsep</code> , <code>format</code> .
<code>\lpzglstsetup{<i>keys</i>}</code>	The same keys, for every list.

continued on the next page

Command	Effect
<code>\lpzgcheck{⟨keys⟩}</code>	Consistency checks on the abbreviations: unknown (on by default), unused , ignore (section 5.4).
<code>\ag. ... \fg.</code>	Glossed sub-example: <code>\a. ... \f.</code> followed by <code>\gll.</code> A shorthand whose name another package already owns is left to it (section 10).
<code>\lpzgadd{⟨abbrs⟩}</code>	Register abbreviations used outside <code>\lpzg</code> so that they reach the list.
<code>\lpzglistentry {⟨abbr⟩}{⟨full form⟩}</code>	One entry of the list; redefine to restyle every list.
<code>\eachwordone/two/three</code>	Font commands for tiers 1–3.
<code>\GlossSep</code>	Glue between gloss columns.
<code>\GlossPhantomAlign</code>	Align gloss words past leading brackets, parentheses and judgment marks (<i>recommended</i> ; section 5.7). Also the package option <code>phantomalign</code> ; <code>\GlossPhantomAlignOff</code> reverts.
<code>\GlossPhantomChars {⟨chars⟩}</code>	Leading characters that alignment skips (default <code>*?#%([<)</code>).
<code>\GlossPhantom {⟨material⟩}</code>	Manual override: pad a gloss word by an invisible box the width of <code>⟨material⟩</code> , in the object font.
<i>References</i>	
<code>\label, \ref</code>	As usual; references print in parentheses.
<code>\sublabel{⟨key⟩}</code>	Label a sub-example for use in a range.
<code>\Next, \NNext, \Last, \LLast</code>	Relative references.
<code>\TextNext</code>	Next example of the running text (from a footnote).
<code>\pref, \pNext, ...</code>	Parenthesis-free twins of all of the above.
<code>\refrange{⟨a⟩}{⟨b⟩}</code>	Compact range, as (3a–c).
<code>\prefrange, \Refrange</code>	Without parentheses; over whole examples.
<i>Further apparatus</i>	
<code>\altn{⟨...⟩}{⟨...⟩}</code>	Stacked alternatives; [<code>⟨c/l/r⟩</code>] alignment. Fallback: <code>\lxAltn</code> .
<code>\altg{⟨alt⟩}{⟨alt⟩}...</code>	Glossed alternatives: written in both lines of <code>\exg.</code> (objects, then glosses); both-braced, centred. Also <code>\lxAltg</code> .
<code>\exsource{⟨text⟩}</code>	Flush-right attribution.
<i>Layout and options</i>	
<code>lazy, gb4e</code>	Package options: dot syntax (default), <code>gb4e</code> environment syntax; combinable.

continued on the next page

Command	Effect
<code>phantomalign</code>	Package option: aligns words in glosses with the word in the object line, not a bracket or judgment mark; combinable.
<code>legacy</code>	Package option: the geometry and dash conventions of <code>linguex</code> ; orthogonal to the syntax options.
<code>\resetExdefaults</code>	Restore all layout lengths to the current mode's defaults.
<code>\firstrefdash</code>	Number–letter dash in “(3a)”; empty by default, – under <code>legacy</code> .
<code>\secondrefdash</code>	Letter–roman dash in “(3a-i)”; – in both modes.
<code>\rangedash</code>	Dash in a whole-example range.

`linguexx` owes its input syntax to Wolfgang Sternefeld's `linguex`, its glossing commands to the `cgloss4e` of Hans-Peter Kolb and Craig Thiersch, the ambition of its gloss engine to John Frampton's `expex`, and its parenthesis-free references to a construction of Alan Munn's. The implementation is its own.