The linguexx package

Numbered examples, interlinear glosses and grammaticality judgments for linguistics

Gerhard Schaden
gerhard.schaden@univ-lille.fr

Version 1.5 — November 3, 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 reimplementation 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 tikz and xspace, plus the expl3 layer, part of the LaTeX kernel since 2020, and it works identically under pdfLaTeX, XeLaTeX 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 **3**

2 Origins, acknowledgments, and relation to other packages **4**

3 Examples **5**
3.1 Starting and ending an example 5
3.2 Sub-examples 6
3.3 Returning to a higher level: `\z.` 6
3.4 Custom labels 7
3.5 An environment interface 8
3.6 The langsci-gb4e interface: `\ea` … `\z` 9
3.7 Examples in footnotes 13

4 Grammaticality judgments **14**
4.1 Automatic marks . . . 14
4.2 Space for judgments . . . 15
4.3 Arbitrary marks: `\jdg` . . . 15

5 Interlinear glosses **16**
5.1 Two and three tiers . . . 16
5.2 Gloss abbreviations: `\lpzg` . . . 16
5.3 The list of abbreviations used: `\lpzglist` . . . 18
5.4 Checking the abbreviations: `\lpzgcheck` . . . 19
5.5 Any number of tiers: `\gl` … `\endgl` . . . 20
5.6 Fonts and spacing . . . 20
5.7 Aligning past brackets and judgment marks . . . 20
5.8 Aligning morphemes . . . 23
5.9 The language of a tier (tagged PDFs) . . . 24
5.10 Right-to-left scripts [experimental] . . . 25
5.11 Abbreviations: `\exg.` and `\ag.` . . . 28
5.12 Variant rows sharing a gloss: `\CommonGloss` . . . 29

6 Cross-references **31**
6.1 Labels . . . 31
6.2 References without parentheses . . . 31
6.3 Relative references . . . 31
6.4 Ranges . . . 32
6.5 Checking what a reference points at: `\exrefcheck` . . . 33

7 Further apparatus **34**
7.1 Stacked alternatives: `\altn` . . . 34
7.2 Glossed alternatives: `\altg` . . . 36
7.3 Source attributions: `\exsource` . . . 38
7.4 A free translation beside the gloss: `\GlossTransSide` . . . 38
7.5 Structural labels: `\exannot` . . . 40
7.6 Movement arrows: `\mvfrom` and `\mvto` . . . 43

8 Example and glossing layout **45**

9 PDF tagging and accessibility **47**
9.1 Spoken forms for tagged PDFs . . . 48
9.2 Stacked alternatives, and the word between them . . . 49
9.3 Gloss abbreviations and their expansions . . . 50
9.4 Transliteration and the engine you compile with . . . 51
9.5 What this does not yet claim . . . 51

10 Notes and limitations **53**

11 For front-end authors **56**
11.1 Protocol A: building a syntax . . . 56
11.2 A worked front-end . . . 57
11.3 The four invariants . . . 58
11.4 Protocol B: a geometry mode . . . 59

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 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). The table of Leipzig glossing abbreviations behind `\lpzg` (section 5.2) follows Natalie Weber's leipzig, to which it owes both the list and the idea of treating the abbreviations as a resource the document can be asked about.

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.

One of them, not both. linguexx replaces these packages rather than accompanying them, and a document must load only one. linguex, expex, gb4e and langsci-gb4e all define `\ex` — as does this one, and all of them with `\def`, so whichever file is read second wins and nothing says so. The result compiles: the examples go on numbering, and what breaks is a detail in the middle of the document. linguexx therefore stops with an error naming the other package, whichever order they were loaded in. The environment and `\ea` syntaxes are available here as the options `gb4e` and `langsci` (sections 3.5 and 3.6), which is what a document that wants them should use. One case cannot be caught this way: expex is plain TEX and may be read with `\input`, which leaves no record of a package having been loaded. There the package warns that `\ex` is no longer its own and leaves the document alone.

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.

A `\ref` to the example being repeated is the usual argument: `\ex.[\ref{ex:hund}]` gives the repetition the number the original carries, whatever that turns out to be. `\exg.` takes the same argument, with one difference in how it is written, described where it is documented (section 5.11).

A custom-labelled example can be labelled and referred to like a numbered one. `\ex.[(7)]\label{ex:seven}` makes `\ref{ex:seven}` print "(7)", and a sub-example of it "(7a)". The brackets are those of the example numbers, so they follow `\ExBareRefs` and `langsci` as a number's do: there, the same reference prints "7". A label written without them, `\ex.[A]`, is referred to as it stands: "A". Each such example has a link target of its own, and `\cref` lists them without ever joining them into a range, since the labels need not follow one another.

A custom label belongs to a whole example, and to nothing smaller: `\ex.` and `\exg.` take one, the sub-example commands `\a.`–`\f.` and `\ag.`–`\fg.` do not. A bracket after one of those is ordinary text, and the letter or roman numeral remains the label — which is what a sub-level is for: its items are identified by their place in it.

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. A third syntax option, `langsci`, adds the `\ea` … `\z` front-end on top of these; it has a section to itself (3.6). These options choose the input *syntax*; a further, orthogonal option `legacy` chooses the *geometry* of the original linguex and may be added to the first two (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 The langsci-gb4e interface: `\ea` … `\z`

langsci-gb4e, Language Science Press's fork of gb4e, writes what the environments of the preceding section write, without the environments. `\ea` opens an example; inside one, it opens the next level down; and `\z` closes whatever the matching `\ea` opened. The depth is never spelt out — it is read off the nesting. The option is `\usepackage[langsci]{linguexx}`, and it implies `gb4e`, so `exe`, `xlist` and `\ex` come with it.

```
\ea LAONE the first example.
\z

\ea LATWO the head of a nest.
  \ea LASUBA the first sub-example.
  \ex[*]{LASUBB judged through the bracket.}
    \ea LAROMAN the roman level.
    \z
  \ex LASUBC back at the letters.
  \z
\ex LATHREE a second top-level example.
\z
```

renders as:

(15) LAONE the first example.

(16) LATWO the head of a nest.
  a. LASUBA the first sub-example.
  b. *LASUBB judged through the bracket.
    i. LAROMAN the roman level.
  c. LASUBC back at the letters.

(17) LATHREE a second top-level example.

Two things in that example are worth naming. `\ex` inside an `\ea` block continues at the level currently open, exactly as it does inside `exe` — so the `\ex` after the inner `\z` is a

second *top-level* example, and the one before it a further letter. And the bracket judgment of `\ex[`*⟨judgment⟩*`]{`*⟨text⟩*`}` is available at every level, `\ea` included: in langsci-gb4e `\ea` expands to `\begin{exe}\ex`, so anything `\ex` accepts, `\ea` accepts.

`\eal` … `\zl` is the second pair: an example whose body is a sub-list, i.e. a number with no text of its own and the letters underneath it.

```
\eal
  \ex LALISTA under a head with no text of its own.
  \ex LALISTB the second letter.
\zl
```

renders as:

(18) a. LALISTA under a head with no text of its own.
b. LALISTB the second letter.

`\eal` is top-level only, as it is in langsci-gb4e; written inside an example it is a package error rather than a guess at what a nested one ought to mean. Where you want a nested list, write `\ea` and open the level below it with a second `\ea`, as in the first example above.

The option also brings langsci-gb4e's deeper gloss stacks, `\glll` through `\gllllllll` — four to eight tiers. They are wrappers over the same engine as `\gll`, and section 5.5 describes `\gl` … `\endgl`, which takes any number of tiers and is available whatever options are in force.

One syntax per example

The dot syntax and the `\ea` front-end may be mixed in one document — that is what `[lazy,langsci]` is for, and the next paragraph is about little else — but not *within one example*. An `\a.` inside an `\ea` example, and an `\ea` inside an `\ex.` example or an `exe` batch, are package errors naming both syntaxes.

The reason is that `\ea` and `\a.` do different amounts of work. `\ea` opens the level, its list *and* its first item in one command; `\a.` opens a level that `\z.` closes. Mixed, the closers no longer match the openers: a `\z.` inside an `\ea` example would close one of the three things `\ea` opened and leave the example half-open, and a `\z` after an `\a.` would close the letter level and leave the letters live for the rest of the document. Both of those typeset without complaint — the first drops the following text into the wrong list, the second silently demotes the next example to a sub-example — so the rule exists to turn a page that is quietly wrong into a message at the line that asked for it.

The mixing described in section 3.5 is untouched: `\a.` inside `exe`, and an `xlist` inside a dot-command example, work exactly as they did. Those two agree about what `\z.` means, which is precisely what `\ea` does not.

A footnote is a new stream, not part of the example it hangs on, so a footnote on an example written one way may hold examples written the other.

One mistake this syntax makes possible and the other two do not is forgetting the closer: an `\ex.` example ends at a blank line and an `exe` batch at its `\end`, but an `\ea` example ends at a `\z` and at nothing else. An `\ea` left open is reported by name, with the line it stood on, and the example is closed so that the rest of the document still typesets — everything after the missing `\z` will have been set inside it. A missing *inner* `\z` has the same effect and the same message: the outer `\z` closes the inner level, and the example itself is what stays open.

Moving a document across, one example at a time

That is what the option is for. Load `[lazy,langsci]` and both syntaxes are live at once:

```
\usepackage[lazy,langsci]{linguexx}
```

Both drive the same engine, so they share one counter, one label system, one set of anchors and all layout parameters. Converting an example therefore renumbers nothing, moves no cross-reference and shifts nothing on the page: a converted example and its unconverted neighbour set their sub-examples at the same indents. The unit of conversion is the whole example, which is also the unit that carries a number and that a `\ref` points at.

One deliberate difference from langsci-gb4e follows from this. There, `\ea` sets its examples ragged right; here it does not, because in a half-converted document that would reflow every converted example against the ones still to come, and a page diff could then no longer tell a mistake from a conversion. `\ExRaggedRight` asks for langsci-gb4e's behaviour; `\eanoraggedright` and `\ealnoraggedright` exist for source compatibility and set justified whatever that switch says.

Finally, `langsci` cannot be combined with `legacy`: `legacy` reproduces linguex's geometry to the value and langsci-gb4e has its own, and a document asking for both has asked for two answers to every length in the package. The combination is a package error rather than a silent choice between them.

The rest of langsci-gb4e

The option carries the rest of the package's surface too, so that a preamble and a document body being ported need no editing. The criterion is what langsci-gb4e *does*, which is what a real source relies on; the list below is accordingly generous. Two names are not provided — see *What this package does not provide* below. Everything here is available under `langsci` and nowhere else.

what	what it does
sub-level numbering	
`\begin{xlista}`	letters, `a.`
`\begin{xlisti}`	lower roman, `i.`
`\begin{xlistn}`	arabic, `1.`
`\begin{xlistA}`	upper alph, `A.`
`\begin{xlistI}`	upper roman, `I.`
items	
`\exi{`*⟨label⟩*`}`	an item with the label given, stepping no counter
`\exr{`*⟨key⟩*`}`	an item labelled with another example's number
`\exp{`*⟨key⟩*`}`	the same, primed
`\sn`	an unnumbered item
the rest of the front-end	
`\eas … \zs`	an example boxed so it will not break across a page
`\eafirst, \zlast,` `\zllast`	the same, without the space above or below
boxes and references	
`\jambox[`*⟨width⟩*`]{`*⟨text⟩*`}`	line *⟨text⟩* up *⟨width⟩* from the right margin
`\jambox*{`*⟨text⟩*`}`	the same, and set \jamwidth to its width
`\attop{`*⟨box⟩*`},` `\atcenter{`*⟨box⟩*`}`	align a box by its top, or on its centre

what	what it does
`\xbox{`*⟨width⟩*`}{`*⟨text⟩*`}`, `\nobreakbox{`*⟨text⟩*`}`	unbreakable boxes
`\xref{`*⟨key⟩*`}`, `\xxref{`*⟨key⟩*`}{`*⟨key⟩*`}`	a reference, and a range in one pair of parentheses
lengths and fonts	
`\exewidth{`*⟨sample⟩*`}`	reserve a label box as wide as *⟨sample⟩*
`\twodigitexamples …`	`(23)`, `(234)`, `(2345)`
`\gblabelsep{`*⟨len⟩*`}`	the gap between label box and text
`\exfont, \exnrfont`	the example body, and its number
`\glossfont, \transfont`	the gloss tiers, and the free translation
`\fnexfont …`	the same four inside a footnote
`\examplesroman, \examplesitalics`	the object tier upright or italic
`\gltoffset, \nogltOffset`	the space above the free translation
`\singlegloss, \nosinglegloss`	set the gloss single-spaced, or not

The four package options come too: `nojambox`, `manualexewidth`, `lowerpenalty` and `nocgloss`. The first three do what they do in langsci-gb4e — suppress `\jambox`, stop the label box from widening by itself once the numbers reach three digits, and let examples break across a page. `nocgloss` is accepted and reports that it has no effect; see below.

Where this differs from langsci-gb4e, and why

Six things are not copies, and each is worth knowing before a page is compared against one set with the original.

- `\exp` keeps *both* meanings. It is the LaTeX kernel's math operator as well as langsci-gb4e's item command, and here the mode decides between them: inside math it is the operator, outside it is the item. A document may use both freely.
- `\exp`'s prime and `\atcenter` are written in *text mode*. langsci-gb4e writes them `$'$` and `$\vcenter{...}$`, and math here would put a `Formula` element in the structure tree, which costs a document its PDF/UA-2 conformance (section 9). `\atcenter` therefore centres on `\ExAtCenterAxis`, half the x-height, rather than on the math axis, which is only readable from inside math mode; against a real `\vcenter` at 10pt the box sits 0.38 pt lower.
- `\eas` boxes its example in a `minipage` and not in a `tabular`, so that the example's prose is not wrapped in a `Table` element — a screen reader would announce it as a table, and PDF/UA does not allow it. A footnote inside one reaches the minipage-footnote numbering noted in section 10.
- `\examplesroman` and `\examplesitalics` set the object tier and not the example font. Setting the object tier is what they are wanted for in a glossed example, and it is what they do here; the example font is `\exfont`'s to set.
- `\subexsep` and `\judgewidth` warn instead of acting. The first sets a label separation for the sub-levels alone, and this package has one `\Exlabelsep` for every level (section 8);

the second sets the width of a reserved judgment column, and here a judgment hangs into the label gutter and reserves nothing — which is exactly why marked and unmarked examples align (section 4.2).

- `nocgloss` is accepted and reports. In langsci-gb4e it stops a bundled copy of cgloss from being read so that another package can own `\gll`; here there is no bundled copy to withhold, and the glossing is what `\exg.`, `\altg`, `\GlossTierLang`, `\lpzg` and the tagged word-bundle structure are all built on. Switching it off would leave each of those needing an answer, and they have more than one; the question is recorded in `doc/DEFERRED-DECISIONS.md` rather than guessed at.

And one difference already noted above: `\ea` does not set its examples ragged right unless `\ExRaggedRight` asks for it.

What this package does *not* provide

`xlistabr` and `qlist` are not provided. They are not silently absent either: both names are defined here, and defined to stop at the line that uses them with a message naming what to write instead — `xlist` for the first, an ordinary LaTeX list for the second. An error at the line that has to change is friendlier than a surprise further down the page.

What `\ref` prints

linguex puts the parentheses in the number: `\theExNo` *is* "(1)", so `\ref` gives "(1)" and prose says "as in `\ref{`*⟨key⟩*`}`". langsci-gb4e puts them in the example's label instead and leaves the counter bare, so `\ref` there gives "1" and prose says "(`\ref{`*⟨key⟩*`}`)" or `\xref{`*⟨key⟩*`}`. Both conventions are in use, and a document carried from the second to the first prints "((1))" on every cross-reference.

So the convention follows the syntax: under `langsci`, `\ref` is bare, at every level and on the footnote series — `1`, `2a`, `2b-i`, `i` — exactly as langsci-gb4e sets them. Without the option nothing changes: `\ref` parenthesises as it always has.

What does *not* follow the convention is everything this package prints itself. The example's own number, `\Next` and its family, `\xref`, `\xxref`, `\exr`, `\exp`, `\refrange` and `\Refrange` spell the parentheses out, so they print the same characters either way; only a plain `\ref`, and the `\cref` that follows it, change. A `langsci` document whose prose already writes "as in `\ref{`*⟨key⟩*`}`" asks for the other convention in one line:

```
\ExParenRefs   % \ref prints (1), as without the option
\ExBareRefs    % \ref prints 1, as langsci-gb4e does
```

Either may be given in the preamble or in the middle of the document; each takes effect from the point it stands at, and the example labels are unaffected by both.

3.7 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]

[1] Like so:

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:

(19) a. Ich habe geschlafen.
b. ?? Ich habe geschlaft.
c. * Ich bin geschlafen.
d. # Ich schlafte.

The mark must come first in the item, *before the example text.* A `\label` or a `\sublabel` may precede it: they make no ink, so the mark is still the first thing on the line and is still hung in the margin.

In a glossed example the mark opens the object line, and it is recognized there however the gloss is written: `\exg.` and `\ag.`, but equally `\gll`, `\glll` and `\gl` after `\ex.`, `\a.`, a head line, or `gb4e`'s `\ex`. It hangs exactly as it does in plain text, and the gloss below starts under the first word, not under the mark.

A star written directly against an opening parenthesis, `*(Er) kommt`, is the notation for an obligatory element. The star is still a judgment and hangs like any other, and the parenthesis stays with its word; only its spoken form differs (section 9.1).

```
\ex. \label{ex:invasione}*?La sola invasione romana della Tracia
```

renders as:

(20) *? La sola invasione romana della Tracia

Anything that does print — a `\textbf`, an opening bracket, a footnote mark — ends the recognition, and a mark behind it stays in the text where it was typed. Use `\jdg` (section 4.3) for those, and for marks outside the four characters below.

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.

(i) First footnote example.

(ii) * Second one, with a judgment.

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

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.

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

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:

(21) † A mark of one's own choosing.

(22) → 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:

(23) †† 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:

(24) 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:

(25) 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 cell in the middle is left empty with `{}`, for a word that has no gloss of its own, such as a gap; the columns after it stay under their own words. 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 9.3. `\lpzg` is self-contained and does not require, or interact with, the leipzig package. It may also stand in a section title: hyperref is told what one means in a PDF string, so the bookmark carries the label as written (`3sg.pst`) instead of the warning and the dropped command a title command it does not know about would earn.

Modified labels. A label, or a piece of one, may be marked up inside the call — to pick the cell of a paradigm the surrounding discussion is about:

```
\gll On-i star-e vladik-e su se posvadjal-i.\\
     those-\lpzg{\textbf{m}.pl} old-\lpzg{f.pl} bishop-\lpzg{f.pl}
       are \lpzg{refl} argued-\lpzg{\textbf{m}.pl}\\
\glt `Those old bishops argued.'
```

renders as:

On-i	star-e	vladik-e	su	se	posvadjal-i.
those-M.PL	old-F.PL	bishop-F.PL	are	REFL	argued-M.PL

'Those old bishops argued.'

The modifier *adds* to the small capitals: what it marks comes out bold *and* small, and the rest of the label is unaffected. That is not automatic. Latin Modern has no bold small capitals in any encoding — there is no bold companion to `lmromancaps` — and plain `\textsc{\textbf{m}}` in such a font loses the shape and keeps the weight, silently: the label comes out as a bold lowercase "m", which is the one thing a category label must not look like. `\lpzg` therefore asks the font, for each piece of the label separately, whether its small capitals are real. Where they are — pdfLaTeX's default `cmr` has them, and so does any OpenType face carrying `smcp` in its bold weight — nothing changes and the label is set with `\textsc` as before. Where they are not, the capitals are made: the letters are set as capitals at `\LpzgCapsScale` (default `0.75`) of the current size, in the font the modifier asked for, so the boldface survives and the height matches the real small capitals beside it. Under active tagging the made capitals carry an `/ActualText` of the letters as written, so a screen reader and copy-and-paste still get `m.pl`; in an untagged document they extract as capitals, as made small capitals always do.

`\textbf`, `\textit`, `\textsl`, `\textup`, `\textmd`, `\textrm`, `\textsf`, `\texttt`, `\textnormal` and `\emph` are recognised inside a label; other commands are left to themselves and their argument stays in whatever small capitals surround it. The markup is invisible to everything else `\lpzg` does: the label is read through it, so `\lpzg{\textbf{m}.pl}` expands, records and lists exactly as `\lpzg{`*⟨m.pl⟩*`}` does.

Labels beyond ASCII. A key may be any text — `\SetLeipzig{`*⟨fém⟩*`}{`*⟨féminin⟩*`}` for a paper written in French, say — and it may be spelt either way its language allows, as the character (`fém`) or as an accent command (`f\'em`). Both are the same key: keys are compared in one normal form, so a label written one way finds a declaration written the other, and `\lpzglist` lists one entry rather than two. That normal form is a byte string, which is why the lists set the key through the same step that reads it back: what you typed is what is printed, on all three engines.

5.3 The list of abbreviations used: `\lpzglist`

A paper that glosses its examples owes the reader a list of the abbreviations it uses. `\lpzglist` 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{`*⟨3sg.pst⟩*`}` contributes three entries — 3, SG and PST — and nothing else. Each entry is set as the abbreviation followed by its full form:

```
\lpzglist
```

renders as:

1	first person
3	third person
ACC	accusative
DEF	definite
DET	determiner
F	feminine
LOC	locative
M	masculine
NOM	nominative
PL	plural
POSS	possessive
PROG	progressive
PRS	present
PST	past
REFL	reflexive
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: linguex 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{`*⟨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 9.3 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

`\lpzglist` takes optional key–value settings for one list; `\lpzglistsetup{`*⟨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 `\lpzgadd`, 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 `\lpzglisttitle` (`\section*` where that exists).
`sep`	Separator of the `inline` style, by default a semicolon.
`itemsep`	Vertical space between entries of the `list` style, `0pt` 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`:

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

`format` is the one-shot form of `\lpzglistentry`, 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 `\lpzglist`: 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
\glt `The dog sleeps.'
```

renders as:

(26)

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:

(27)

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:

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

The padding is an invisible box the width 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 `( [ <`. A judgment mark that opens the example's object line is not among them: it is a judgment, and hangs in the margin (section 4); the marks counted here are those on a word further in. Change the set with `\GlossPhantomChars{`⟨*marks*⟩`}`, e.g. `\GlossPhantomChars{[(}` to react to square and round brackets only.

A translation usually keeps the delimiters of what it translates, and such a gloss word is padded only by the *difference*. In

```
{\GlossPhantomAlign
\ex. \gll della sua morte (\`e questa)\\
of.the his death (is this.one)\\
}
```

renders as:

(29) della sua morte (è questa)
of.the his death (is this.one)

the gloss carries a parenthesis of its own, so its first real glyph already sits where it belongs and nothing is added: the two parentheses stay flush. Were the tier set at a smaller size its parenthesis would be narrower than the object's, and the pad would be exactly that difference — enough to put the real glyphs under one another, and no more. Should the gloss open with *more* mark than the object word does, aligning the real glyphs would mean pulling the gloss to the *left* of its column, into the space belonging to the word before it; that is not done. The gloss keeps the column and its extra mark hangs out on the left.

When the gloss's marks are the *beginning* of the object word's run rather than its end — `(` under `([`, or under a parenthesised Quine corner as below — they stay under the marks they reproduce, and the pad goes between them and the word: the two parentheses are flush, and the first real glyphs still line up.

The argument is a list without separators: *one token is one mark*, so a character and a command count alike and neither commas nor spaces are needed between them. That is how `\#` and `\%` are in the default set at all — a bare `#` is a macro parameter character and a bare `%` opens a comment, so neither can be typed in an example in the first place, and the escaped forms are the only ones that exist. It is also how you add a mark of your own:

```
\GlossPhantomChars{*?([<\#\%\dag}
```

leaves the default set alone and adds `\dag` to it. A command that is not in the set is not a mark: it stays in the word, where you typed it.

There is one restriction on what may be a mark, and it is worth stating as a rule: *math-mode code typed directly into the example cannot be a delimiter; a command that contains math can.* A Quine corner written `$\ulcorner{}$` in the object word cannot be declared, however you spell the declaration. The same corner given a name — `\qq`, defined as `\ensuremath{\ulcorner}` — can be, and from then on behaves like every other mark in the set.

The reason is *one token is one mark* read the other way round. The scan looks at the first *token* of the word, and in `($\ulcorner{}$machin` that token is the `$`, which is not a mark; the scan stops there, before it has so much as seen `\ulcorner`. So putting `\ulcorner` in the set changes nothing, and says nothing — the gloss simply goes on being padded as though the corner were part of the word. `\ensuremath{\ulcorner}` written out in the example fails identically, its first token being `\ensuremath`. Nor is declaring the `$` itself a way round it: the scan would then peel the dollar and the corner off and stop at the empty group between them, leaving a leading run with its math unbalanced — `! Extra }, or forgotten $`, at the end of the example. A name is the only thing here with a single token to offer:

```
\newcommand{\qq}{\ensuremath{\ulcorner}}
\GlossPhantomChars{*?([<\#\%\qq}

\ex. \gll AAA (\qq{}machin BBB\\
     aaa \qq{}thingy bbb\\
```

The empty group after `\qq` does two jobs. It keeps the control word off the letters that follow it without using a *space* — a space there would end the word, and `\gll` would build two columns out of the one — and the scan stops at a brace group, so the leading run comes out as exactly `(\qq` and no more.

Without the declaration that example still lines up, as it happens: the corner is then simply part of the word, in the object line and in the gloss alike, and the parenthesis alone accounts for the difference between them. Declaring it earns its keep when the gloss does *not* carry the corner — then `thingy` sits under `machin`, where undeclared it would sit under the corner.

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`, and it is also what makes a judgment mark hang to the left of a stack instead of displacing the alternative it marks — in `\altn` (section 7.1), and in `\altg` on the tier that carries the object language (section 7.2).

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:

(30) 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 Aligning morphemes

A column of a gloss is aligned by word: the gloss starts under the object word, and nothing inside the two lines up. In a heavily segmented example that leaves the reader to count hyphens. `\GlossMorphAlign` aligns the morphemes as well, so that each one starts at the same point in every tier that is segmented, its hyphen included:

```
\GlossMorphAlign
\ex. \gll ev-ler-imiz-de otur-uyor-uz \\
          house-\lpzg{pl}-\lpzg{1pl.poss}-\lpzg{loc}
          sit-\lpzg{prog}-\lpzg{1pl} \\
\glt `We are sitting in our houses.'
```

renders as:

(31) ev -ler-imiz -de otur-uyor -uz
house-PL-1PL.POSS-LOC sit -PROG-1PL
'We are sitting in our houses.'

There is no markup to learn: a word is split at the boundaries a source written to the Leipzig rules already has, `-` between affixes and `=` before a clitic. Each segment is as wide as the widest segment in its position, so in the tier that holds the widest one the segments simply follow each other, and the alignment shows in the other tiers.

- *Braces keep a unit whole.* A gloss of several words that the author writes with hyphens, `{in-law}-mother`, is two segments, not three; a word braced as a whole is one.
- *A delimiter with nothing on one side is not a boundary.* A citation form such as `-t` and an en dash such as `1914--1918` are left alone.
- *A tier that does not split is set whole.* It does not take part, which is what a three-tier gloss with an unsegmented first line needs; and `schlaf-t` over `sleep.3sg` is simply a segmented word over a gloss with no boundary.
- *Segmented tiers must agree.* If two of them split one column into different numbers of morphemes — usually a typo, since the Leipzig rules ask for the counts to match — that column is aligned by word, and the log names it: "Gloss column 2: the segmented tiers split into 3 and 4 morphemes, which do not match". The rest of the gloss is still aligned.

`\GlossMorphAlign` is a declaration, off by default, and obeys the usual scope rule: in the preamble it covers the document, inside a group a single example, and `\GlossMorphAlignOff` turns it off again. It works with the `phantomalign` padding of section 5.7, which counts into the first morpheme of the tier it pads. A column that holds an `\altg` or a movement arrow (`\mvfrom`, `\mvto`) is aligned by word: finding the widths means typesetting each morpheme once more, and those two are commands that must run exactly once. Under tagging nothing changes: the morphemes are placed within the word they belong to, and the structure tree of an aligned gloss is that of the same gloss aligned by word.

5.9 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.

An empty code takes a declaration back: after `\GlossTierLang{2}{}` tier 2 has no language of its own again, under the same scope rule (inside an example, for that example only). Such a tier is read in the language of the document — the one given to `\DocumentMetadata` — because a PDF element without a language takes that of what contains it; nothing needs to be written for it.

The free translation is almost always in the language of the glosses — which, in a paper written in one language and glossing into another, is not the language of the document. So a translation takes the language of the gloss's *last* tier, when that tier has one declared: a French paper glossing into English says so once, and its translations are read as English too.

```
\GlossTierLang{1}{de}          % object line is German
\GlossTierLang{2}{en}          % glosses -- and so translations -- English
```

The last tier and only that one: when it declares nothing, as in the common case where only the object line has a language, the translation has no language of its own and is read in the document's, exactly as an undeclared tier is. It never borrows the language of the object line, or of a transliteration line above the glosses. A `\gl` whose last line is not a gloss — a line of categories, say — therefore passes nothing on.

`\GlossTransLang{`*⟨code⟩*`}` sets the translation's language directly, and wins over the gloss's; `\GlossTransLang{}` lets translations follow the glosses again. It obeys the usual scope rule, like `\GlossTierLang`:

```
\GlossTransLang{en}            % translations English, whatever the glosses
```

Either way the language reaches the structure tree only through these declarations: babel's `\foreignlanguage` does *not*, so a translation in a language that neither its gloss nor `\GlossTransLang` declares 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.

5.10 Right-to-left scripts [experimental]

NB Consider the current implementation of examples in right-to-left scripts (and especially, the right-to-left grid) an experimental feature, that may change. If you have strong convictions and/or good ideas on how this should look like, please do not hesitate to reach out to me.

Data in Hebrew, Arabic, Persian, Urdu or Yiddish can be glossed in two ways, and linguexx supports both: the native script as a line of its own above an ordinary gloss of a transliteration, which is what most linguistic publications do, or a gloss whose grid itself runs from right to left, with the native script aligned word by word. Both need LuaLaTeX (recommended) or XeLaTeX; pdfLaTeX cannot set right-to-left text properly, and both commands below stop there with an error that says so. The font is the author's, as for any tier: load one with fontspec and give it to the tier with `\GlossTierFont`.

A line in the native script. `\glscript{`*⟨lang⟩*`}{`*⟨text⟩*`}` sets *⟨text⟩* as one unaligned line, in its own direction, above the gloss that follows; under tagging it carries *⟨lang⟩* as its `/Lang`:

```
\newfontfamily\hebfont{FreeSerif.otf}[Script=Hebrew]  % fontspec
\GlossTierLang{1}{he-Latn}
\ex. \glscript{he}{\hebfont הילד אכל את התפוח}
\gll ha-yeled axal et ha-tapuax \\
     \lpzg{def}-boy ate \lpzg{acc} \lpzg{def}-apple \\
\glt `The boy ate the apple.'
```

renders as:

(32) הילד אכל את התפוח
ha-yeled axal et ha-tapuax
DEF-boy ate ACC DEF-apple
'The boy ate the apple.'

`\glscript` works for a left-to-right script too — a Greek or Cyrillic line over a transliteration — and then under pdfLaTeX as well.

A right-to-left grid. `\GlossRTL` sets the following glosses from right to left: the first word stands in the rightmost column, the columns run leftwards, and a gloss too long for the line goes on from the right. Which tiers are written right to left is not declared a second time: it is read off `\GlossTierLang`. A language written right to left — `he`, `ar`, `fa`, `ur`, `yi`, `ps`, `syr` and a few others — makes its tier right-to-left, and a script subtag decides when there is one, so `he-Latn`, a transliteration, is left to right and `az-Arab` right to left. The other tiers keep their own direction inside their columns:

```
\GlossTierLang{1}{he}
\GlossTierLang{2}{he-Latn}
\GlossTierFont{1}{\hebfont}
\GlossRTL
\ex. \glll הילד אכל את התפוח \\
           ha-yeled axal et ha-tapuax \\
           \lpzg{def}-boy ate \lpzg{acc} \lpzg{def}-apple \\
\glt `The boy ate the apple.'
```

renders as:

(33) הילד אכל את התפוח
ha-tapuax et axal ha-yeled
DEF-apple ACC ate DEF-boy
'The boy ate the apple.'

`\GlossRTL` is a declaration with the usual scope; `\GlossRTLOff` returns to left-to-right grids. The example number, the free translation and anything else around the grid stay where they are in a left-to-right document. A grid that fits on one line sits next to its number, as a left-to-right one does; to find out whether it fits, the gloss is set once without being shown, so a grid containing an `\altg` or a movement arrow — commands that must run exactly once — is placed against the right margin instead.

Under tagging nothing about the order changes: the engine reverses the columns on the page, while the structure tree keeps the words in reading order, first word first, each tier in its own language. XeTeX writes right-to-left text into the PDF in visual order, so there each right-to-left word also carries its text as `/ActualText`; with LuaTeX none is needed. This is one reason to prefer LuaLaTeX; the other is that XeLaTeX refuses a movement arrow inside a right-to-left grid, because it would reverse the drawing of the arrow along with the text. `\exannot` cannot go on a right-to-left gloss, and `\GlossTransSide` gives way to it, with a warning, and puts the translation underneath.

Two behaviours are provisional, and recorded as such in the package's notes on deferred decisions: a judgment mark hangs on the left, where it does in every example, although the word it judges is at the right; and under `\GlossMorphAlign` (section 5.8) a morpheme sits at the end of its box, so that every word ends at its column's edge, and a hyphen closes the morpheme before it rather than opening the next one (`ha-` `yeled` over `DEF-` `boy`). Either may change.

A right-to-left document. Everything above is about right-to-left material in a left-to-right document. When the document's *main* language is written right to left — polyglossia's `\setmainlanguage{hebrew}`, under LuaLaTeX or XeLaTeX — examples are the mirror image of what they are in a left-to-right document: the number stands against the right margin, the text the same distance in from it, a sub-example's letter at the text's edge, and a judgment mark to the right of what it judges. Braces open on the right. This needs polyglossia: with babel's right-to-left support (babelbidi included), linguexx sets every example as in a left-to-right document and says so in a warning.

Examples follow the direction of the document's main language, whatever the language of the example. An example inside `\begin{english}` in a Hebrew document is laid out like every other example there — numbered on the right, letters and marks on the right — and only its words run left to right; a short line stands against the text, next to its letter. A Hebrew example in an English document is the same the other way round. A label reads in the document's direction too, since it belongs to the example's frame and not to its text: in a Hebrew document a sub-example is ".a" as well as alef with its period on the left, the period standing between the letter and the text as it does in "a. text"; in an English document a Hebrew example's letter is alef with its period on the right.

Sub-examples are lettered as the main language letters them (alef, bet, … in Hebrew); `\let\Exalph\ExLatinAlph` gives a, b instead (section 8). Under LuaLaTeX, Latin words inside a right-to-left example are set right to left, letter by letter, as they are in polyglossia's own `\texthebrew`: put them in `\textenglish`.

A gloss runs in the direction of its example's language, the language being glossed. In a Hebrew document a Hebrew example's grid runs from the right, as everything there does; an English example's grid, inside `\begin{english}`, runs from the left, and is the mirror image of what `\GlossRTL` makes of a Hebrew example in an English document: a short grid stands beside its number, against the text edge, and a long one starts every line at the left margin. The translation is in the document's language and direction, the reader's, as it is under `\GlossRTL`. The other way round, a Hebrew example in an English document is glossed as `\GlossRTL` glosses, without `\GlossRTL` being written. Nothing needs declaring for either: in such an example a tier whose language is not declared with `\GlossTierLang` is read as the example's language on the object line and as the document's on the gloss lines — English glossed in Hebrew, or Hebrew glossed in English — and a declared language still decides.

A complete document, an English example glossed in Hebrew and a Hebrew one glossed in English, for LuaLaTeX or XeLaTeX, shown with the Hebrew as an editor with bidirectional support shows it:

```
\documentclass{article}
\usepackage{fontspec}
\setmainfont{FreeSerif.otf}
\usepackage{polyglossia}
\setmainlanguage{hebrew}
\setotherlanguage{english}
\newfontfamily\hebrewfont{FreeSerif.otf}[Script=Hebrew]
\usepackage{linguexx}
\pagestyle{empty}
\begin{document}
:באנגלית, הפועל העזר נושא את הזמן

\begin{english}
\ex. \gll The boy-s have eat-en the apple-s\\
          ה- ילד-רבים עזר אכל-בינוני ה- תפוח-רבים\\
```

```
\glt 'הילדים אכלו את התפוחים.'
\end{english}

\ex. \gll הילדים אכלו את התפוחים\\
          the.boy-PL eat.PST-3PL ACC the.apple-PL\\
\glt 'הילדים אכלו את התפוחים.'

-כמו ב\LLast\ -וב\Last.
\end{document}
```

renders as:

באנגלית, הפועל העזר נושא את הזמן:

(1) The boy-s have eat-en the apple-s
ה- ילד-רבים עזר אכל-בינוני ה- תפוח-רבים
'הילדים אכלו את התפוחים.'

(2) הילדים אכלו את התפוחים
the.boy-PL eat.PST-3PL ACC the.apple-PL
'הילדים אכלו את התפוחים.'

כמו ב-(1) וב-(2).

The first example is numbered at the right margin; its English words run from the left, each Hebrew gloss under its word, and its translation reads right to left like the rest of the page. The second is an ordinary Hebrew example, its grid running from the right. The file and its output come with this manual, as `rtl-hebrew.tex` and `rtl-hebrew.pdf`.

Tagged documents. For a tagged document with right-to-left text, LuaLaTeX is more than a preference. Under XeLaTeX, every paragraph written right to left fails PDF/UA-2, with or without linguexx; a gloss passes, but a head line or a translation in Hebrew does not (section 9.5).

5.11 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:

(34) *Das Beispiel funktionieren
the example work.INF
(intended: 'The example works.')

So does the custom label of section 3.4, which is what repeats a glossed example under the number it had:

```
\exg.[\ref{ex:hund}] Der Hund bellte. \\
     the dog  bark.PST               \\
```

With one difference from `\ex.`: the bracket must follow `\exg.` immediately. A bracket that a *space* separates from it is the first word of the object line — `[` is one of the openers `\GlossPhantomChars` lists, and phantom alignment (section 5.7) exists to hang it in the gutter — so `\exg. [DP der Hund] bellte.` glosses a bracketed constituent and takes a number of its own. `\ex.` has no object line for a bracket to open and follows the LaTeX convention of skipping the space.

5.12 Variant rows sharing a gloss: `\CommonGloss`

Sub-examples that differ only in the order of the same words can share one gloss. In an example that contains `\CommonGloss`, each of `\ag.` through `\fg.` reads a single line — a *variant row* — and each `\CommonGloss` line is set under all of them, column by column. A row with no word in a column has `{}` there:

```
\ex. Haitian
\ag. \textit{Bay} kiyès li achte liv la {} ? \exannot{P-pattern}\\
\bg. *{} kiyès li achte liv la \textit{bay} ? \exannot{V-pattern}\\
\CommonGloss give who \lpzg{3sg} buy book \lpzg{det} give\\
\glt `Who did s/he buy the book for?'
```

renders as:

(35) Haitian

a.		*Bay*	kiyès	li	achte	liv	la	?	P-pattern
b.	*		kiyès	li	achte	liv	la	*bay* ?	V-pattern
	give	who	3SG	buy	book	DET	give		

'Who did s/he buy the book for?'

- *Rows and lines.* Up to six variant rows, `\ag.` to `\fg.`, each ended by `\\`, then any number of `\CommonGloss` lines, then `\glt` as usual. The rows must have the same number of cells — a word, a braced group, or `{}` for a gap — since a forgotten gap would shift every later word into the wrong column; a shared line may be shorter (its last cells stay empty) but not longer. Both are errors rather than guesses.
- *Only here.* `\ag.` reads one line only in an example that has a `\CommonGloss`; in every other example it reads its object and gloss lines as in section 5.11. A dot-syntax example is read ahead, which is how it knows; an `exe` batch is not, and announces the table instead (below). In a `langsci \ea` a shared gloss is an error.
- *Sub-examples.* The rows are the sub-examples a, b, …: a `\label` after `\bg.` refers to (1b), and a judgment that opens a row — `* ? \# \%`, `\jdg`, or a mark made with `\DeclareJudgment` — hangs beside the row's letter exactly as in an ordinary sub-example, also when the row's first cell is a gap.
- *Annotations.* An `\exannot` at the end of a row goes into a column of its own, at `\ExAnnotColumn` (section 7.5), or `\ExAnnotSep` past the table if the table reaches that far. `\ExAnnotFit` does not measure a shared gloss.

- *Alignment, fonts, languages.* `\GlossPhantomAlign` and `\GlossMorphAlign` work as in an ordinary gloss. With several rows, phantom alignment starts the first real letter of every word in a column past the widest run of leading marks in that column, so rows with and without a bracket line up with each other as well as with the gloss. For `\GlossTierFont` and `\GlossTierLang`, the variant rows are tier 1 and the n-th `\CommonGloss` line is tier $n + 1$.

- *Width.* A shared gloss is not broken across lines; one wider than the line is an error. Split it, or give the variants glosses of their own.

- *Direction.* In a right-to-left document, and for an example whose language runs against the document's, the table is laid out as an ordinary gloss in the same place (section 5.10): the letters, marks and annotation column on the side of the document's direction, the columns in the example's. A Hebrew example in a Hebrew document is the mirror image of the one above; an English one there has its letters on the right and its columns from the left; a Hebrew one in an English document has its letters on the left and its first column on the right. Unlike an ordinary right-to-left gloss, such a table may carry `\exannot`: its annotation column belongs to the frame.

- *Slides.* In a beamer frame, a `\pause` between the rows uncovers them one at a time, as between ordinary sub-examples: row a comes with the shared lines on the first slide, each further row on the next. A `\pause` among the shared lines delays the lines after it. A row not shown yet keeps its space, so nothing moves from slide to slide.

In an `exe` batch (option `gb4e`) the rows are the items of an `xlist` that says so with the option `[CommonGloss]`: each `\ex` (or `\item`) reads one row up to its `\\`, and `\ex[`*⟨judgment⟩*`]` gives the row a judgment the `gb4e` way. The table is set exactly as the dot syntax sets it, and an example with no head line puts its number on row a's line:

```
\begin{exe}
\ex Haitian
\begin{xlist}[CommonGloss]
\ex \textit{Bay} kiyès li achte liv la {} ? \exannot{P-pattern}\\
\ex[*] {} kiyès li achte liv la \textit{bay} ? \exannot{V-pattern}\\
\CommonGloss give who \lpzg{3sg} buy book \lpzg{det} give\\
\glt `Who did s/he buy the book for?'
\end{xlist}
\end{exe}
```

The list must be the example's own sub-level, not one inside another `xlist`, and must end with at least one `\CommonGloss` line; its rows are not set without one. As with any `\ex[`*⟨…⟩*`]`, a row whose first cell starts with `[` is written `\ex[] [...`. An `xlist` without the option is unchanged.

In a tagged PDF a shared gloss is a *table*, not a gloss: each row is a table row whose header is its letter and judgment, and each word is a cell, gaps included. A screen reader reads the variants and the shared lines one after the other, and can move down a column to hear which word glosses which — where an ordinary gloss, read column by column, would interleave the variants word by word.

6 Cross-references

6.1 Labels

Examples and sub-examples are ordinary LaTeX 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:

(36) a. Erste.
b. Zweite.

See (36), and in particular (36b).

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:

(36b) with parentheses, 36b without.

6.3 Relative references

To refer to a neighbouring example without labelling it:

Command	Refers to
`\Next`	the next example
`\NNext`	the one after that
`\Last`	the previous example
`\LLast`	the one before that
`\TextNext`	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.

Each of them, and each `p`-twin, takes an optional sub-example part: `\Last[`*⟨letter⟩*`]` refers to that letter of the example `\Last` resolves to and prints it inside the parentheses, "(3b)". The separator is `\firstrefdash`, the same hook `\ref` uses for a `\sublabel`, so the two spell an example alike — "(3b)" by default and "(3-b)" under `legacy`. Nothing verifies that the letter exists; it is text you supply, as it is in linguex.

Clickable references. With hyperref loaded, each of these — and each p-twin — prints as a link to the example it names, exactly as the `\ref` it abbreviates does. No label is needed: every example gets a destination whether or not anyone asks for one, from hyperref where it makes them and from linguexx where it does not. beamer is the case where it does not — it switches hyperref's implicit anchors off and runs its own navigation — and the links work there too. On a frame with overlays the link lands on the slide where the example *appears*: an example after a `\pause`, or inside `\uncover` or `\only`, is anchored where the reader can see it rather than where TEX first ran it.

A reference that names an example the document does not have — `\LLast` before the second example, `\Next` after the last one — prints its number as it always did and is deliberately *not* linked. A link to a destination that does not exist is not an error; the reader would simply be taken somewhere plausible and wrong. Instead every such reference is named, with its input line, in one warning at the end of the run:

```
Package linguexx Warning: No example carries the number a relative
(linguexx)                reference asks for: 0 (line 12), 9 (line 40).
```

A number that *two* examples carry is withheld in the same way and reported in its own words. That happens when the example counter is reset — with `\setcounter`, or by `legacy`'s per-chapter reset in a class that has chapters — because the anchor hyperref attaches to an example is built from the counter alone, so the two examples claim one anchor and hyperref keeps only the first. A `\ref` to the second has always led to the first for the same reason; what the links promise is that they add no second wrong jump to a document that has that one.

Which examples exist is read from the `.aux`, so the links are a run behind: a document whose examples have just moved is asked to rerun rather than told that its references dangle. The optional part is printed but not aimed at — `\Last[`⟨*b*⟩`]` links to the example, not to its letter b — because that letter is text you typed and need not name a sub-example that exists. Pass `norelreflinks` to the package to print the numbers without links, as linguex does.

The space after the command. These are control words, so TEX's tokenizer discards the space that follows one before any package can see it. Written plainly, `\Last shows` would therefore set as "(1)shows". Each of them ends in `\xspace` — as linguex's references do — which puts the space back before a word and leaves it out before punctuation, so `\Last shows` and `\Last, but` both come out right and neither needs guarding. The `\Last\ ` and `\Last{}` that documents written against the older behaviour contain stay correct too: `\xspace` recognises both and adds nothing.

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. Being clickable makes a reference that has drifted easier to notice, since it now lands on the wrong example rather than merely naming it, but no less likely. 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:

(37) a. Erste.
b. Zweite.
c. Dritte.

The examples in (37a–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`, and `lxCustomSerial` for custom-labelled examples (section 3.4); in a list that mixes the two kinds, the custom-labelled ones come after the numbered ones.

`\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.

6.5 Checking what a reference points at: `\exrefcheck`

A relative reference is positional. `\Next` is the example after the point where it stands, so when an example is inserted between the prose and the example it was written for, `\Next` quietly moves to the new one. Nothing is wrong on the page: the number is correct for where the reference now stands, and nothing anywhere says it is not the example that was meant. `\exrefcheck` lists, at the end of the run, every reference this package resolved, with the *opening words* of the example it reached, so that a wrong one can be seen as wrong:

```
\exrefcheck              % six words of each target
\exrefcheck[words=10]    % or as many as you like
```

The list goes to the terminal and the `.log`; nothing is typeset, and the document comes out exactly as it does without the command. For a `\Next` written before an example that was added later, it reads:

```
l.24, p.1: \Next -> (1) on p.1: "INSERTED une phrase ajoutée après coup, ..."
```

— the line and page of the reference, the number it printed, the page the example is on, and the example's own words. It covers `\Last`, `\Next`, `\NNext`, `\LLast` and `\TextNext` with their `\p…` twins (listed under the plain name), and both ends of `\refrange`, `\prefrange` and `\Refrange`. It does not cover `\ref`, which belongs to the kernel and to whichever of hyperref, cleveref or varioref you load, and is left alone for that reason.

The command can go in the preamble or anywhere in the document. Like `\ref` it reads what the previous run recorded, so its first run says that it needs another one; it also says so when the examples have changed since the run it is reporting on.

Where the words come from depends on the syntax. A dot-syntax example (`\ex.`) is read in whole before it is set, and is quoted from that, its `\label`s left out. A gloss is quoted by its first line, in any syntax, and gb4e's `\ex[`*⟨judgment⟩*`]{`*⟨text⟩*`}` by its argument. A plain `\ex` item in an `exe` environment, or one opened by `\ea`, is set straight from the input and is not read ahead of, so the list gives its number and page and says that its text was not recorded.

Under beamer, a reference is listed once however many slides its frame has, and the pages given are the slides on which the reference and its example can be *seen*: material after `\pause` is set on the frame's earlier slides too, only invisibly, and those are not counted.

A relative reference finds its example by position, so it is listed correctly even after the example counter has been reset. A range is given by labels and resolves by number, so a range end whose number two examples carry is reported as such rather than guessed at.

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:

(38) {~~This girl in the red coat~~ / She / Mary} will put a picture of {Bill / him} 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 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.

Judgment marks in a stack. Alternatives that differ only in their acceptability are the ordinary case for a stack, and the mark that says so should not push the word it judges out of line:

```
\ex. La secrétaire de Jean et collaboratrice de Paul
     \altn[l]{est}{*sont} à la gare.
```

Left-aligned and taken literally, that stacks `est` above `*sont`, which puts `est` under the star and `sont` a star-width to its right — so the one pair of words the reader is asked to compare is the one pair that does not line up. With `\GlossPhantomAlign` in force (or the package option `phantomalign`; section 5.7) the mark is set in a narrow column of its own, hanging to the left of the alternatives and flush against them, and the words line up:

renders as:

(39) La secrétaire de Jean et collaboratrice de Paul $\left\{\begin{matrix}\text{est}\\ \text{*sont}\end{matrix}\right\}$ à la gare.

The mark stays *inside* the braces, because it judges its own alternative and not the stack. The marks that count are those of `\GlossPhantomChars`, and where several rows carry marks of different widths the column takes the widest, so every mark sits flush against the words.

Nothing is inserted between a mark and the word it marks: that distance is the font's own, exactly what you get by typing `*sont` in running text, and alignment does not change it. What moves is the *other* rows, which come right to meet the marked one — so an aligned stack is no wider than the same stack unaligned.

A stack that carries a mark also tucks `\AltJdgTuck` (default `0.2em`) closer to its opening brace. The room is there: a brace's arm curls *away* from the content between its tips, so at the height of any row its ink sits well inside the box, and the mark moves into that hollow without touching it. The effect is to undo part of what the hanging costs: at 10pt on a one-star stack the opening brace sits 7.7pt from the unmarked rows instead of 9.7pt, while the mark keeps 2.3pt of air. The default is chosen so that the air holds at every size — 2.3pt from 10pt to 20pt — which a deeper tuck does not, since the brace's pen grows with the font and `\AltBraceAmplitude` does not. Only the opening gap changes, so the closing brace rides along with the stack and the distance from the preceding word is untouched; and only a stack that actually carries a mark tucks. Set `\AltJdgTuck` to `0pt` for the untucked layout, or deeper than the default to trade the mark's air for words that stay where an unjudged stack puts them.

Like the other brace lengths, it is not clamped: a large enough value pulls the stack straight through the brace, and TEX has no opinion about overlapping ink, so nothing would otherwise say so. Past `\AltBraceWidth` + `\AltBraceSep` the package therefore writes one warning to the log, naming the value and the threshold, and *uses the value anyway* — a setting that silently did something other than what it was set to would be worse than one that obeys. The warning is reported once per document, and only for a stack that actually carries a mark. A stack in which no alternative carries a mark is set

exactly as it would be with the option off, so turning the option on never moves a stack that has nothing to align.

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 drawn braces, so no mathematics is involved (the package needs tikz, not amsmath). The brace is drawn as a filled outline rather than a stroke, because a typographic brace does not have one width everywhere: like the `{` of a maths font it carries its weight in the spine and tapers at the two ends and at the tip. The braces are tunable through seven macros, redefined with `\renewcommand`. Three 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; `\AltBraceAmplitude` (`4pt`) is 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; and `\AltBracePen` (`0.11em`) is the weight of the spine — the figure is Computer Modern's own, whose brace stem measures `1.20pt` at 11 pt, so the default puts the brace at the colour of the type around it. The other four govern placement, not shape: `\AltBraceSep` (`0.15em`, the gap between brace and stack), `\AltBraceOuterSep` (`0.35em`, 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, by about `1.1pt` at 11 pt), `\AltBraceRaise` (`0pt`, a vertical nudge), and `\AltJdgTuck` (`0.2em`, how far a stack carrying a hanging judgment tucks toward its opening brace — see above). `\altg` (section 7.2) draws its braces with the same `\AltBraceWidth`, `\AltBraceAmplitude` and `\AltBracePen`, so changing any of them 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:

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

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.

Punctuation after a paradigm. A paradigm is one block spanning both tiers, and its closing brace is drawn by the gloss call — half a line lower and a gloss column further right than the object call. A sentence-final period, though, is written where a period is written: right after the object `\altg` that ends the object line. Punctuation glued to the object call (no space between) therefore travels with the paradigm and is set after the closing brace, level with the middle of it — the sentence it ends is the paradigm's, not the first alternative's:

renders as:

(41) On-i su se {posvadjal-i argued.M.PL; *posvadjal-e argued.F.PL}.
they are REFL

Typeset where it stands it would land immediately after the object stack — inside the braces, in the gutter between the two columns, reading as a period on the first alternative. It keeps `\AltBraceSep` from the brace, the clearance the brace keeps from the stack on its other side: the tip points straight at the punctuation at this height, and a period flush against it reads as part of the brace. Anything separated from the call by a space is a gloss column of its own and needs none of this.

Judgment marks: the object tier only. With `\GlossPhantomAlign` in force (or the package option `phantomalign`; section 5.7), a judgment mark on an object alternative hangs to the left of the stack instead of pushing the word it marks out of line, exactly as in `\altn` (section 7.1):

renders as:

(42) Die {Frau woman.SG; *Socke sock.SG; Maus mouse.SG} ist hier.
The.SG is.PRS here.

The gloss stack follows the widened object stack of its own accord, so the paradigm stays a two-column block. The *gloss* tier is deliberately left out: a judgment is a claim about the object language, and a gloss is a translation of that language rather than something that is itself grammatical or not, so a mark typed in a gloss alternative stays where you put it. If you want a mark to govern a whole paradigm rather than one alternative, put it on the example with `\jdg`.

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`, `\AltBracePen`, `\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:

(43) 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`.

7.4 A free translation beside the gloss: `\GlossTransSide`

By default `\glt` sets the free translation under the interlinear grid. `\GlossTransSide` puts it in a column beside the grid instead, which on a slide buys the thing there is least of:

```
{\GlossTransSide
\ex. \gll il mio libro\\
          the my book\\
     \glt `my book, and a translation with room of its own'
}
```

renders as:

(44)
il	mio	libro	'my book, and a translation
the	my	book	with room of its own'

It is a declaration and it respects grouping, like `\ExAnnotFit` and `\ExRaggedRight`; `\GlossTransBelow` turns it off again. `\glt` itself is unchanged — it still takes no argument, and no document that does not ask for the side position is affected in any way.

The declaration has to come *before* the example, and that is not a style preference. By the time `\glt` is reached the grid is already a finished paragraph contributed to the

enclosing list, and there is nothing left to set beside it; the decision has to be made while the grid is still being built.

The share of the measure the grid gets is `\GlossTransRatio` (default `.6`), and the gap between the two columns is `\GlossTransSep` (default `2em`). Both are ordinary parameters, settable for one example or for the whole document. `\GlossTransRightSkip` (default `0pt plus 2em`) is a little stretch at the right of the translation's column: a narrow measure hands TEX lines it cannot break any shorter, and a fully justified column then overflows rather than ending short. All three defaults follow expex, so that one example set with either package comes out in much the same proportions. The gloss is what is being read closely, and it gets the room.

Two restrictions, each a package error rather than a silent reinterpretation:

- **Top-level examples only.** Below the top level the measure has already been reduced twice, so both columns come out narrow and the grid starts wrapping between columns; and a split taken from the indented measure would put every sibling's translation at a different place. The restriction is cheap to lift if a document turns up wanting it — `doc/EXPEX-GAPS.md` records what such a design would have to answer.
- **No `\exannot` on the same gloss.** Its column is measured from `\columnwidth`, which says nothing once the grid has been narrowed to half of it.

If there is no room — a narrow measure, a `twocolumn` paper — the translation goes underneath after all and the log says so. The threshold is `\GlossTransMinWidth` (default `6em`), measured on the translation's column.

A side-by-side gloss is one unbreakable block: the two columns are boxes, and a box does not break across a page. A tall one therefore wants a page with room for it, and says so as an overfull `\vbox` if it does not get one.

Where it earns its keep is a gloss that already runs to several lines. This is expex's own illustration of the same feature, from §12.2 of its manual (printed p. 52), reproduced here so that the two can be compared on one example:

```
{\GlossTransSide
\renewcommand\GlossTransRatio{.69}  % this gloss wants the room
\newcommand\gc[1]{\textsc{#1}}
\ex. \gll Homâo sa čô pô tha ñu nao ngã hmua. Ñu djă gă, ñu djă
          čŏng ñu, laih gui rêo ñu. Todang bboi rôk jolan ñu nao
          hma, ñu bbôh sa droi mrã dõ bboi gah, a, hruh ñu.\\
          \gc{exist} one \gc{clf} person old \gc{3s} go do field
          \gc{3s} hold machete \gc{3s} hold hoe \gc{3s} and
          carry.on.back back.basket \gc{3s} while at along trail
          \gc{3s} go field \gc{3s} see one \gc{clf} peacock stay
          at \gc{drct} -- nest \gc{3s}\\
     \glt `There was an old person who went to work in the field.
           He took along his machete, he took along his hoe, and he
           carried his basket on his back. While he was on his way
           to the farm, he saw a peacock beside its nest.'
}
```

renders as:

(45) Homâo sa čô pô tha ñu nao ngă hmua. Ñu
EXIST one CLF person old 3S go do field 3S
djă gă, ñu djă čŏng ñu, laih gui
hold machete 3S hold hoe 3S and carry.on.back
rêo ñu. Todang bboi rôk jolan ñu nao
back.basket 3S while at along trail 3S go
hma, ñu bbôh sa droi mră dŏ bboi gah,
field 3S see one CLF peacock stay at DRCT
a, hruh ñu.
– nest 3S

'There was an old person who went to work in the field. He took along his machete, he took along his hoe, and he carried his basket on his back. While he was on his way to the farm, he saw a peacock beside its nest.'

`\GlossTransRatio` is raised to `.69` here, which is what the parameter is for. A gloss of nine words to a row wants width more than running prose does, so the `.6` default leaves this one cramped, while too generous a share leaves the translation's last line a single word. There is no formula: try two or three values and look at the result. That is a fair description of the whole feature — it is a typesetting decision, and the parameter is there so that one example can make it differently from the rest of the document.

When the side position actually saves space. Less often than one would think, and it is worth being plain about. Narrowing the measure makes *both* columns taller, so the arrangement pays only when one of them would have been much shorter than the other — the short one then costs nothing, because it hides inside the tall one's height. Measured on this package's own examples:

translation	underneath	beside	
one line	71.1pt	57.6pt	saves a line
three lines	84.7pt	94.1pt	costs most of one

(Measured on a two-tier gloss of three words in a 25 mm-margin article; the point is the direction, not the millimetre.)

So: reach for it when the translation is short beside a gloss of some height, which is the common case in a slide, and leave it alone when the translation is the longer of the two. The Jarai example above is worth setting this way not because it saves height — it saves about four points, a quarter of a line — but because the translation stays *beside* the gloss it belongs to instead of arriving after eight lines of it. `\gc` is a local shorthand for this example only: expex writes `\\{exist}` for it, which is not available here because `\\` separates the gloss tiers. In a linguexx document the Leipzig abbreviations would be `\lpzg{`*`clf`*`}` and so on, which sets the small caps *and* gives a screen reader the expansion to speak (section 5.2); they are left plain here only to keep the example the one expex prints, since three of its glosses are not Leipzig abbreviations at all.

A long gloss in a narrowed column may of course wrap between columns, as one in a full-width measure would; the side position makes that more likely rather than less. If it happens, the vertical space the arrangement was supposed to buy is what pays for it, and `\GlossTransBelow` for that example is the answer.

7.5 Structural labels: `\exannot`

`\exsource` puts its argument at the right margin, which is where a source belongs. A label *about* an example — the category of a constituent, the reading intended, the language —

belongs beside the example instead, and beside several examples it has to form a column, or the eye cannot use it. `\exannot` does that:

```
\setlength{\ExAnnotColumn}{9cm}
\ex.
\a. que Pierre est fatigu\'e\exannot{[CP]}
\b. Pierre est fatigu\'e\exannot{[TP]}
\z.
```

renders as:

(46) a. que Pierre est fatigué [CP]
b. Pierre est fatigué [TP]

`\ExAnnotColumn` is where the column is, measured from the *left* edge of the text block — so "9cm" means what it says on the page, and does not move when the example does. The default is `.75\columnwidth`. `\ExAnnotSep` (default `1em`) is the least gap between an example and the column; give it no stretch or shrink. The font is `\ExAnnotFont`, `\normalfont` by default.

The column holds across nesting levels, because it is measured from the text block and not from the indented line; and it holds however long the examples are, because an example that would come within `\ExAnnotSep` of the column takes its annotation onto the next line rather than pushing the column right:

renders as:

(47) a. a short one [CP]
b. one long enough that its annotation has to go somewhere else [TP]
i. and one a level deeper [DP]

`\exannot` closes the example's paragraph, as it must in order to hold the last line to the column, so it comes last in its example.

In a gloss, write it at the end of the *object* line — with a space in front of it or glued to the last word, whichever reads better. It is set level with the object tier, at the same column as everywhere else — not under the free translation, and not as a column of the grid:

```
\ex. \gll que Pierre est fatigu\'e \exannot{[CP]}\\
          that Pierre is tired\\
     \glt `that Pierre is tired'
```

renders as:

(48) que Pierre est fatigué [CP]
that Pierre is tired
'that Pierre is tired'

On any other tier it is an error, and so is one in the middle of a line: a gloss is a translation, so a label on a gloss tier has nothing to align with, and one between two object words has nowhere to go.

What a screen reader says. "`[CP]`" is an abbreviation a reader cannot expand and a synthesiser will not usefully spell. Give `\exannot` a spoken form and, under tagging, the annotation reaches the structure tree as a `Span` carrying `/Alt`: the page still shows `[CP]`, copy-and-paste still yields `[CP]`, and the reader hears the phrase. Either per annotation, or once for a label used throughout:

```
\SetAnnotSpoken{[CP]}{complementizer phrase}
\ex. que Pierre est fatigu\'e\exannot{[CP]}          % from the table
\ex. Pierre est fatigu\'e\exannot[tense phrase]{[TP]} % just this one
```

This is the same arrangement as `\DeclareJudgment[spoken=]` and `\SetJudgmentSpoken` (§4 if you have met those already). An annotation with no spoken form gets no `Span` at all: an `/Alt` that repeats the text it replaces is noise in the tree.

`/Alt` is used rather than `/E` or `/ActualText` because it is the one of the three that leaves both the printed label and the text layer alone; `doc/TAGGING-NOTES.md` sets out the comparison for anyone auditing a document's structure.

Letting the examples choose the column: `\ExAnnotFit`. `\ExAnnotColumn` is a number you have to find, and the number that looks right depends on the examples underneath it. `\ExAnnotFit` finds it instead. Each annotated example records where its text ended, and the column of an example — one `\ex.` and everything under it — is put `\ExAnnotSep` past the longest of them:

```
\ExAnnotFit
\ex.
\a. que Pierre est fatigu\'e depuis mardi\exannot{[CP]}
\b. Pierre est fatigu\'e\exannot{[TP]}
\z.
\ex.
\a. short\exannot{[DP]}
\b. brief\exannot{[VP]}
\z.
```

renders as:

(49) a. que Pierre est fatigué depuis mardi [CP]
b. Pierre est fatigué [TP]

(50) a. short [DP]
b. brief [VP]

The unit is the *example*, not the document: the second block above gets a column of its own, because its examples are shorter. Footnote examples are their own examples in this sense too.

The widest *annotation* of a block counts as well, and the column is never narrower than it. That is what keeps a long label in line with its neighbours instead of letting it step out to the left on its own:

renders as:

(51) a. que Pierre est fatigué [CP]
b. Pierre est fatigué [an annotation wider than that column]
c. Pierre dort [TP]

When the two pull against each other — a label so wide that the column has to move left of where some example's text ends — the column wins and that one annotation goes onto the next line. It is the column that has to hold; a line is only a line.

This costs an `.aux` round trip, so a fresh document's first run sets `\ExAnnotColumn` and says

```
Package linguexx Warning: Annotation columns
are not settled. Rerun to get
\ExAnnotFit right
```

The second run is right, and silent. If the warning comes back on a later run, something is still moving: the one way that happens is an example whose own text breaks across lines, where the annotation is part of what the paragraph breaker is fitting. Set `\ExAnnotColumn` for that block and turn the fitting off inside it (`\ExAnnotNoFit`); both are ordinary switches and respect TEX grouping.

Nothing is recorded and no round trip happens unless `\ExAnnotFit` is asked for.

And `\jambox`. Under `langsci` this package also provides Alexis Dimitriadis's `\jambox` (§3.6), which answers a neighbouring question: it lines material up a fixed distance from the *right* margin, and for a note that belongs near the margin it is the more direct instrument of the two. `\exannot` is aimed at the other case — a column set close to the examples, where what has to be held is the column's position rather than its distance from the margin — and the two are built differently because of it. `\jambox` is provided exactly as langsci-gb4e has it; use whichever suits the note you are writing.

7.6 Movement arrows: `\mvfrom` and `\mvto`

A movement has two ends: the position a constituent started in, and the one it landed in. Mark each with the same label, and linguexx draws the arrow from the first to the second:

```
\ex. \mvto{a}{What} did John buy \mvfrom{a}{\_\_}?
```

renders as:

(52) What did John buy ___?

`\mvfrom[`*⟨options⟩*`]{`*⟨label⟩*`}{`*⟨text⟩*`}` marks the base position and `\mvto[`*⟨options⟩*`]{`*⟨label⟩*`}{`*⟨text⟩*`}` the landing site; the *⟨text⟩* is typeset as it is, so a copy can be greyed out with `\textcolor` as usual. The arrow always points at the `\mvto`, whichever of the two comes first in the source, so rightward movement is written the same way. A label belongs to its example — to one `\a.` or `\b.` — so `a` can be used again in the next one. A mark with no partner in its example draws nothing and says so in the log.

Several movements in one example are stacked out of each other's way: an arrow over a stretch that holds another one runs a level further out, and the line is given the room it needs, so the arrows never reach the next example. A word can be both ends at once, which is how a chain is written; the arrow arriving and the one leaving then meet it side by side, and since the two only share that word, the links of a chain stay on one level:

```
\ex. \mvto{b}{What} do you think \mvto{a}{\mvfrom{b}{\_\_}}
     John bought \mvfrom{a}{\_\_}?
```

renders as:

(53) What do you think ___ John bought ___?

Arrows go below the line, and above the object line in a gloss, where below would cut through the gloss tier:

```
\ex. \gll \mvto{a}{Was} hat Hans \mvfrom{a}{\_\_} gekauft?\\
          what has Hans \_\_ bought\\
     \glt `What did Hans buy?'
```

renders as:

(54) Was hat Hans ___ gekauft?
what has Hans ___ bought
'What did Hans buy?'

The options go on either mark; where the two disagree, the one written second wins. `above` and `below` choose the side, `level=`⟨*n*⟩ puts the arrow ⟨*n*⟩ steps out instead of where the stacking would, and `spoken` is described below. The look of the arrow is the TikZ style `lx move`, so

```
\tikzset{lx move/.append style={red, dashed}}
```

changes every arrow of the document, or of a group. `\MoveArrowGap` (`.3ex`) is the gap between a word and the end of its arrow, and `\MoveArrowStep` (`1.1ex`) the distance from one level to the next — and from the outermost arrow to the line beyond it, which is kept as clear of it as a further level would be; both are macros, so `\renewcommand` them.

Two runs. Where the other end of an arrow is on the page is only known once the page has been written out, so the arrows are drawn from what the previous run recorded, as a `\ref` is. A document compiled once has no arrows yet, and the log asks for a rerun whenever they have moved; `latexmk` does this by itself. Two runs are always enough, however many overlay slides a frame has: each slide records its own positions, so arrows in a beamer frame stay with their words when material appears above them.

One line. An arrow joins two words on one line. If a line break separates them, the arrow is still drawn, but the room for it is kept on the second line only, and the log says so; across a page break nothing is drawn. In a long example, a `\linebreak` before the first marked word keeps the two together.

Screen readers. In a tagged PDF the arrow itself is decoration — an artifact — and what it means is said by the words at its ends. The base position is read with a note after it, "___ (moved)", while the page still prints what you wrote; the landing site is read as printed. `\SetMoveSpoken{`⟨*word*⟩`}` replaces the note — `\SetMoveSpoken {`⟨*déplacé*⟩`}` for a French document — and respects grouping, like `\SetAltSpoken`. The `spoken` option replaces the whole spoken form of one mark, on either end: `\mvfrom[spoken=trace of what] {`⟨*a*⟩`}{`⟨*___*⟩`}`.

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.

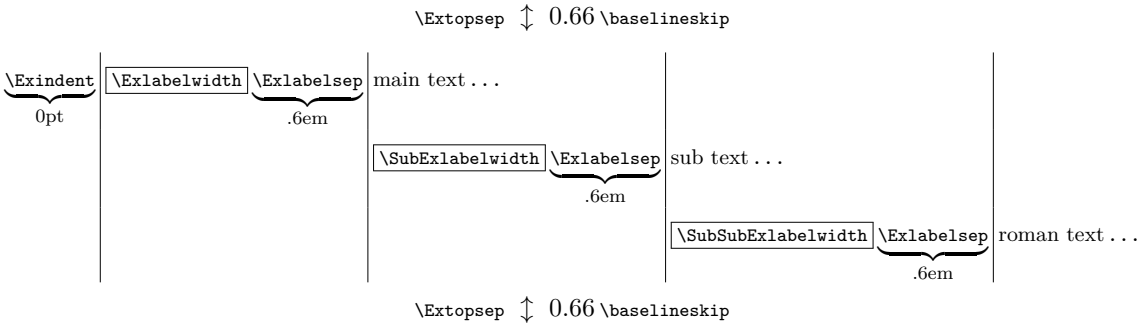


Figure 1: The default box model. On each line, reading left to right: an optional `\Exindent` (0 pt), the label box, then the shared gap `\Exlabelsep`; the text of a level begins one label box plus one `\Exlabelsep` to the right of the level above, so the boxes cascade down the page. Each box is named for the length that fixes its width (`\Exlabelwidth` 2.6 em, `\SubExlabelwidth` 1.6 em, `\SubSubExlabelwidth` 1.6 em) 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
`\Extopsep`	`.66\baselineskip`	Vertical space before and after an example.
`\Exredux`	`-.66\baselineskip`	Correction inserted between *consecutive* examples, so the gap is not doubled. Keep it at `−\Extopsep` for an even rhythm.
`\Exlabelwidth`	`2.6em`	Width of the number box — room for "(199)". Widen it for four-digit numbering.
`\Exlabelsep`	`.6em`	Gap between any label box and its text.
`\SubExlabelwidth`	`1.6em`	Width of the letter box ("a."). Deliberately wider than the letter: the surplus is where a hanging judgment goes (section 4.2).

continued on the next page

Length	Default	Role
`\SubSubExlabelwidth`	`1.6em`	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 `\SubExlabelwidth`, so the two sub-levels share one box width.
`\JdgSep`	`0.15em`	Gap between a hanging judgment and the text.
`\ExAnnotColumn`	`.75\columnwidth`	Where `\exannot`'s column is, measured from the *left* edge of the text block, so it does not move with the nesting level (section 7.5).
`\GlossTransSep`	`2em`	Gap between the gloss and a side translation (section 7.4); `\GlossTransRatio` (`.6`, a factor) is the share of the measure the gloss gets, `\GlossTransMinWidth` (`6em`) the width below which the side position is abandoned, and `\GlossTransRightSkip` (`0pt plus 2em`) the stretch at the right of the translation's column.
`\ExAnnotSep`	`1em`	Least gap between an example and that column. Give it no stretch or shrink: it is what makes an example that comes within it take its annotation onto the next line rather than crowd in. Under `\ExAnnotFit` the same length is the distance the fitted column keeps past the longest example of its block.

A few further knobs are macros, not lengths, and are changed with `\renewcommand`: `\GlossSep` (glue between gloss columns), `\ExAnnotFont` (default `\normalfont`) 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 number of a top-level example is bracketed by `\ExLBr`/`\ExRBr`, and a footnote example's by `\FnExLBr`/`\FnExRBr`. These are linguex's names and linguex's meaning: `\renewcommand{\ExLBr}{[}` together with `\renewcommand{\ExRBr}{]}` prints "[1]" in the label and in every reference to it, and the footnote series keeps its parentheses until its own pair is moved as well. The package reads all four through `\theExLBr` and its relatives, which add the parenthesis suppression that `\ExBareRefs` asks for (section 6); redefine one

of *those* and you replace the switch along with the character.

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

The sub-example letters, in labels and in references alike, are `\Exalph{`*⟨counter⟩*`}`, and the numerals of the level below are `\Exroman{`*⟨counter⟩*`}` (linguex's names again). By default they are `\alph` and `\roman`, so they follow the document's language: under polyglossia's Hebrew the sub-examples are lettered alef, bet, and so on, in Hebrew script. For Latin letters whatever the language, say

```
\let\Exalph\ExLatinAlph
```

in the preamble, or anywhere from then on. `\alph` itself cannot do this, because a language package that changes the letters changes what `\alph` produces. Any other style is a matter of defining `\Exalph` as a one-argument command that takes the counter's name and expands to the letter.

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 LaTeX project's PDF tagging code. If the document begins with a `\DocumentMetadata` line that turns tagging on, for example, on a current 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 LaTeX. The `tagging=on` key is the recommended form from the 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 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 5.9), so it is pronounced with its own phonetics; and category abbreviations written with `\lpzg` carry their expansion (section 9.3), 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. Under XeLaTeX that holds only for a document without right-to-left text (section 9.5). 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.

A caution on stability. Tagging in LaTeX requires a recent LaTeX. 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
`*`	ungrammatical
`*(`	obligatory
`?`	questionable
`??`	highly questionable
`?*`	extremely degraded
`#`	infelicitous
`%`	grammatical for some speakers

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

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

`*(` is not a mark of its own: it is the spoken form of a lone star written against an opening parenthesis, as in `*(Er) kommt`, where the parenthesis marks an element as obligatory and "ungrammatical" would say the opposite. `\SetJudgmentSpoken{*(}{...}` changes it like any other.

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 Stacked alternatives, and the word between them

A stack set by `\altn` or `\altg` is a vertical arrangement on the page, and a screen reader cannot see that the rows are alternatives rather than lines. Under active tagging linguex therefore gives the stack an alternate text that says so in words: `\altn{`*⟨aa⟩*`}{`*⟨bb⟩*`}` is announced "aa or bb", and three alternatives become "aa, bb, or cc".

That word is English, and so is the punctuation convention around it — English puts a comma before the final connector and most other languages do not. `\SetAltSpoken{`*⟨word⟩*`}` replaces the connector:

```
\SetAltSpoken{ou}      % "aa ou bb",   "aa, bb ou cc"
\SetAltSpoken{oder}    % "aa oder bb", "aa, bb oder cc"
```

Write the word bare, without spaces: the spacing around it is the package's business, so `\SetAltSpoken{ou}` and `\SetAltSpoken{ ou }` are the same setting. Dropping the comma before the connector is part of the same change, because that is what these languages do.

To keep it — the English convention, and the package's default — use the starred form. An optional argument replaces the punctuation itself:

```
\SetAltSpoken*{or}     % "aa, bb, or cc"   <- the default, spelled out
\SetAltSpoken{y}[;]    % "aa; bb y cc"
\SetAltSpoken{}        % "aa bb"           <- no connector at all
```

The setting respects grouping, like the xlist variants and unlike `\SetLeipzig`, so one example may be spoken in another language without disturbing the rest of the document:

```
{\SetAltSpoken{ou} \ex. \altn{aa}{bb} }
```

It applies to `\altg` as well as `\altn`, and as with everything in this section it changes nothing on the page and nothing at all in a document compiled without active tagging.

9.3 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

label	meaning	label	meaning
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		

9.4 Transliteration and the engine you compile with

One trap is worth knowing about, because nothing warns you and the page looks perfect. Under pdfLaTeX, characters whose diacritic sits *below* the letter are not single glyphs: TeX 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	pdfLaTeX extracts	XeLaTeX/LuaLaTeX extract
`kṛṣṇaḥ` (Indic)	`kr.s.n.ah.`	`kṛṣṇaḥ`
`ḍāṇṭaḥ` (Indic)	`d . ān.t.ah.`	`ḍāṇṭaḥ`
`ģimeņu` (Latvian)	`gimen ,u`	`ģimeņu`

This manual is itself built with LuaLaTeX, for exactly this reason: it contains those characters in the table above, and under pdfLaTeX 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 ṃ`) 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 XeLaTeX or LuaLaTeX, 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 č ő ł ø þ` and the rest — are unaffected and extract correctly under all three engines.

9.5 What this does not yet claim

The structure this package writes is validated, and validation is not the same as knowing that a document reads well. It is worth being explicit about the distance between the two.

No screen reader has been in the room. veraPDF can say that a file conforms to PDF/UA-2. It cannot say that a reader using JAWS, NVDA, VoiceOver or Orca gets a good account of an interlinear gloss. Nothing here has been tried with one. The structural decisions are argued rather than observed — above all the decision to read a gloss word bundle by word bundle, so that an object word and its gloss are heard together, instead of the whole object line and then the whole gloss line. That is the choice this package would defend, but it is a choice, and only a user can settle it. Treat this part of the package as a well-formed proof of concept: correct by every check that can be automated, and not yet confirmed by the only check that counts.

The spoken forms are English. Three things carry words rather than markup, and all three are written in English:

- the judgment phrases — "ungrammatical", "questionable" and the rest (section 9.1);
- the Leipzig expansions — "singular", "accusative" and some two hundred others (section 5.2);
- the word `\altn` puts between stacked alternatives when it builds their spoken form: "A, B, or C".

The first two can be replaced entry by entry, with `\SetJudgmentSpoken` and `\SetLeipzig`, and the third with `\SetAltSpoken` (section 9.2).

So setting `lang=fr` in `\DocumentMetadata` is not enough to make a French document read properly. The tree would then declare French while its `/Alt` and `/E` strings remained English words, and a French voice would pronounce them as French. Nor is there a way to mark them as English in passing: the `Span` that carries an `/Alt` does not also carry a `/Lang`, so the phrase is announced in whatever language surrounds it.

Calling that a translation job would be too quick, because for the gloss abbreviations it is not clear what should be translated. The printed label is already free: `\lpzg` small-caps whatever you type and never consults the table, and keys are compared with their accents normalised, so `\lpzg{fém}` prints FÉM today, and `\SetLeipzig{f\'em}{féminin}` gives it an expansion although it spells the key the other way. A French document can be glossed in French, entry by entry, right now. What is undecided is what the package should *ship*: whether the Leipzig abbreviations stay in place in every document — they are English-derived but serve as an interchange standard well beyond English — with only the spoken expansions translated, or whether a language should have its own labels as well. The first keeps a glossed example legible to any reader of the field, and costs the link between the label and the word spoken for it. The second is more coherent inside one document, and gives up that portability. This manual does not answer the question, and the package does not answer it by default; `doc/DEFERRED-DECISIONS.md` records what would settle it.

Right-to-left text under XeLaTeX. A tagged document compiled with XeLaTeX fails PDF/UA-2 wherever a paragraph is written right to left: veraPDF finds text that is neither tagged nor marked as an artifact (rule 8.2.2-1). The cause is outside this package. A document with one Hebrew word and no linguexx fails in the same way, because XeTeX reverses a right-to-left stretch of a line together with the tagging markers inside it, so that the markers end up empty and the text after them untagged. LuaLaTeX is not affected.

linguexx keeps its own markers out of the reversal, so its interlinear glosses and `\CommonGloss` tables pass under both engines, and their words and labels read in logical

order. The parts of an example that are ordinary text do not: a right-to-left head line, a free translation in Hebrew or Arabic, a sub-example without a gloss. These fail as any such paragraph does, and so does the document's own right-to-left text. For a tagged document that contains right-to-left text, use LuaLaTeX. The printed page is correct under both engines, and an untagged document is unaffected.

The kernel's tagging is still provisional. These features sit on the LaTeX Project's tagging code at `phase-III`, whose interfaces are declared provisional by the project itself. Where this package leans on an internal it does so behind a guard that degrades to no tagging rather than to a broken document, but the arrangement will want revisiting as that work moves out of testphase.

What is checked, and how often. One accessible configuration is validated with veraPDF on every run of the test suite, and the worked accessible document shipped in `examples/` before a release. The number of combinations a document can present — syntax option, gloss shape, beamer, footnotes, a language per tier — is much larger than the set under validation. Support for `/E` and `/Alt` also varies between readers; a reader that ignores `/E` will simply say "sg".

If you use this with a screen reader, reports of what it actually sounds like are very welcome. That is the missing evidence, and no amount of validation substitutes for it.

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.
- **Verbatim in an example body.** A dot-syntax body is *collected* before it is typeset — that is what lets a blank line end it and `\z.` pop a level — and a command that reads its argument verbatim has to set its catcodes *before* the characters are read. The collector therefore reads the argument of three such commands itself, and hands it over at typesetting time: `\verb` (and `\verb*`), fancyvrb's `\Verb`, and listings' `\lstinline`, with their stars, options and delimiters, including `\lstinline`'s `{...}`. They work as they do anywhere else, in sub-examples and as words of a gloss:

  ```
  \ex. The bound variable \verb|x_1| is written \Verb*|x 1| in the source.
  ```

  renders as:

  (55) The bound variable `x_1` is written `x␣1` in the source.

  Two things still cannot work. An example inside another command's argument — `\footnote{⟨…⟩}`, a beamer frame without `[fragile]` — cannot hold verbatim text, exactly as no `\verb` can in an argument; linguex says so in an error. And no other verbatim command is read specially: `verbatim`, `lstlisting`, fancyvrb's `Verbatim`, short

verbatim made with `\MakeShortVerb`, `\DefineShortVerb` or `\lstMakeShortInline`, and minted's `\mintinline` fail in a dot-syntax body, with whatever error their first special character causes — often "Missing $ inserted".

For those, use the environment syntax, which does *not* collect, so that any verbatim command works there as in running text, including inside an `\a.` or an `xlist` written within the batch. The same holds for `langsci`'s `\ea` … `\z`:

```
\MakeShortVerb{\|}  % shortvrb
\begin{exe}
\ex A body with |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: no verbatim command works there.

- **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 none of linguex, gb4e and langsci-gb4e settles either: all three number such examples on the main series too, so a document arriving from any of them keeps the numbers it had. Nothing is claimed here and nothing is patched; if the numbering matters in your document, put the examples outside the minipage.

- **A stack inside a stack, under tagging.** `\altn` may be written inside an alternative of another `\altn`, and the page is right: both stacks are drawn and braced, the inner one set inside the outer's row. What is not right is the *spoken* form. The outer stack's alternative text is built from the source of its alternatives, so a nested call contributes its own bracketed option and its alternatives run together with no connector between them; and because the inner stack is built before the outer's element is opened, the two come out as siblings rather than one inside the other, so a screen reader reaches the inner stack first and announces its alternatives twice. Nothing is claimed here and nothing is patched: what a nested stack ought to *say* is undecided, and guessing would turn the guess into the promise. If the example is meant to be read aloud, write it as two examples.

- **Sub-examples across a page break, under tagging, before LaTeX 2026-06-01.** With an older kernel, a nested list that breaks across a page in a tagged document can end the run with "`The number of automatic begin … and end …`" from tagpdf, and every structure element after the break is attached to the root: the page looks right, and veraPDF fails the file. Sub-examples are nested lists, so they are affected, but so is the kernel's own nested `enumerate` without linguexx: it is the kernel's bug, and LaTeX 2026-06-01 (TeX Live 2026) fixes it. Not every break point triggers it. With an older kernel, keep such an example on one page, for instance with a `\pagebreak` or `\enlargethispage` before it.

- **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 "ç" — 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.
- **`\altn` has no gloss tier.** It stacks bare constituents; alternatives that each need a gloss of their own are `\altg`'s, written once in the object line and once in the gloss line (section 7.2).
- **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 For front-end authors

Nothing in this section is needed to write a document. It is for someone building a *package* on top of linguexx: a third input syntax, or a geometry to sit beside `legacy`.

The two syntaxes this package ships — the dot commands and the `exe`/`xlist` environments — are thin. Between them they are a few dozen lines; everything underneath, the numbering, the label boxes, the judgment machinery, the glossing and the PDF tagging, is shared and knows nothing about which syntax called it. That seam is exposed under documented names, so a further syntax can be built without touching internals and without a fork.

The names below are the contract. Everything else in `linguexx.sty` — anything spelled `\lx@…` or `\__lx_…` — is private and may change in any release. Public names are expl3 functions, so a front-end calls them under `\ExplSyntaxOn`; they exist whatever combination of `lazy`, `gb4e` and `legacy` the document asked for.

The contract runs the other way too. A release may *add* public names, and does not otherwise touch them: an existing `\lx_…` loses its meaning or its name only in a major version, and only after the changelog has announced it as deprecated in a release before that. A front-end pinned to a major version therefore keeps compiling.

11.1 Protocol A: building a syntax

Function	Effect
Lifecycle	
`\lx_example_begin:`	Open an example: reset the sub-level depth and counters, and save the cross-referencing state.
`\lx_example_end:`	Close it: every sub-level still open, then the main list, then the paragraph bookkeeping.
The body	
`\lx_body_collect:N` ⟨*function*⟩	Collect the example body under this package's terminator rules — a blank line, `\z.`, or an unmatched `\end` — and hand it to ⟨*function*⟩, which must take one argument and be `\long`. Optional: a syntax whose body is delimited some other way simply does not call it.
Items	
`\lx_item_main:`	Step the counter, open the main list, and make the item. For an example that brings its own list.
`\lx_item_main:n {`⟨*label*⟩`}`	The same with a label of your choosing, and no counter step.
`\lx_item_next:`	Step the counter and make the item *without* opening a list: the next example in a batch.
`\lx_item_core:`	Step the counter of whatever level is open and set its label; emit nothing.
`\lx_item_emit:`	Emit the item for the current label, and hang any judgment collected since the last scan.

continued on the next page

Function	Effect
`\lx_item_judged:n` {⟨*mark*⟩}	Core, mark and item in one move, at the current level.
Sub-levels	
`\lx_sub_push:`	Open a new, deeper level and give it its first item. Two levels maximum.
`\lx_sub_next:`	The next item at the level now open.
`\lx_subenv_begin:`	Open a deeper level *without* an item, for a level bounded by an environment: the environment's own group undoes the depth advance at its `\end`.
Judgments	
`\lx_judgment_scan:`	Peek for judgment marks in the input, then emit the item. This is where the `lx_item_` commands end.
`\lx_judgment_scan:n` {⟨*code*⟩}	Peek for marks, then run ⟨*code*⟩.
`\lx_judgment_set:n` {⟨*mark*⟩}	Set the mark explicitly, for a syntax that takes it as an argument. Any mark is allowed, not only the scanned set.
Lists	
`\lx_list_main_open:`	Open the main example list by hand, for the batch shape.
`\lx_label_guess:`	Stand in the label the *next* item will print. Required before opening a list that has no item yet.
`\lx_list_open:n` {⟨*decl*⟩} `\lx_list_close:`	The list funnel (see invariant 4).
`\lx_ol_class:n` {⟨*class*⟩}	The `/ListNumbering` attribute class for the list about to open.
Tagging	
`\lx_tag_if_active:TF`	Whether tagging is switched on *and* the kernel has the commands for it. The one guard.
`\lx_tag_span:nn` {⟨*keyvals*⟩} {⟨*content*⟩}	Set ⟨*content*⟩ inside a tagged `Span`. Without active tagging the content is set bare, so untagged output is unaffected.
`\lx_tag_span_open:n` `\lx_tag_span_close:`	The outer half alone, for content that carries marked content of its own.
`\lx_tag_span_begin:n` `\lx_tag_span_end:`	The inner half as well, for leaf content.

11.2 A worked front-end

The whole of a dot-syntax-shaped front-end. It collects a body the way `\ex.` does, and gives it one item:

```
\ExplSyntaxOn
\NewDocumentCommand \pex { } { \lx_body_collect:N \__fe_run:n }
\cs_new_protected:Npn \__fe_run:n #1
  {
    \lx_example_begin:
    \lx_item_main:
    #1
    \lx_example_end:
  }
\NewDocumentCommand \pa { } { \lx_sub_push: }
\NewDocumentCommand \pb { } { \lx_sub_next: }
\ExplSyntaxOff
```

That is enough for `\pex`, `\pa` and `\pb` to number, to nest, to take judgments typed into the text, to be cross-referenced, to carry glosses, and to tag themselves. The batch shape — one list holding several examples, as `exe` does — is the same parts in a different order: `\lx_example_begin:`, then `\lx_label_guess:` and `\lx_list_main_open:` to open the list, then `\lx_item_next:` per example, then `\lx_example_end:`.

A fuller version of both, exercising every function in the table, is `tests/frontend.tex` in the distribution. It is a regression case: it is written against the public names only, so it stops compiling if one of them goes away, and `veraPDF` is run on what it produces.

Glossing is deliberately outside Protocol A, and its absence from the table is not an invitation to rebuild it: the interlinear tagging — the column spans, the object-language `/Lang`, the abbreviation `/E` — is what makes a gloss PDF/UA-valid, and reassembling it from the raw span helpers is work nobody should repeat. A front-end that wants a gloss syntax of its own should translate it onto the user-level engine, `\gl` … `\endgl` (section 5), which takes any number of tiers and styles them per tier through `\GlossTierFont` and `\GlossTierLang`; the tagging travels with it. A gloss protocol of the same standing as Protocol A is possible later, once there are real front-ends to shape what it should expose.

11.3 The four invariants

These are not style advice but constraints the engine relies on. Each can be broken without any error being raised, and the output then looks plausible, which is what makes them worth stating.

1. **The label is fixed before the list opens.** Under `legacy` the width of the label box is measured from the label itself, so a list opened first is sized from whatever the previous example left behind. This renders identically in the default mode, which is what makes it easy to write and hard to see: use `\lx_item_core:` before `\lx_list_main_open:`, never after.

2. **The grouping structure is the depth stack.** `\lx_sub_push:` advances the depth inside the group it opens for the new level, so leaving that group pops it and `\lx_example_end:` can close whatever is left. Never advance or reset the depth yourself. linguex tracked it in a global counter, and the drift that followed — an example that quietly starts one level deep because an earlier one ended badly — is the thing this design makes impossible.

3. **No extra group around a whole example.** It is tempting to wrap the lifecycle in `\begingroup`…`\endgroup` for safety. Do not: the tagged-PDF "block" code accounts for text units across the example, and an enclosing group desynchronises that

accounting for an example inside a footnote. What the group would have protected is restored explicitly by `\lx_example_end:`.

4. **Every list goes through the funnel.** Use `\lx_list_open:n` and `\lx_list_close:`, not `\list` or `{list}` directly. The funnel uses the environment form, which is the interface the tagging code supports — part of its state restoration is keyed to hooks that command-form `\list` never fires — and it is where the `/ListNumbering` attribute is attached. A list opened around it loses both.

11.4 Protocol B: a geometry mode

A mode is a defaults table and three geometry hooks, and nothing else: `legacy` differs from the default in exactly those four macros.

```
\ExplSyntaxOn
\lx_mode_new:nn {name}
  { defaults = \myDefaults, main = \myMain,
    sub = \mySub, subsub = \mySubSub }
\lx_mode_select:n {name}
\ExplSyntaxOff
```

The values are macro *names*, not bodies, so that you may define them with plain `\newcommand` in ordinary catcodes — a defaults table contains glue whose spaces matter (`.5em plus .3em`), and those would be stripped from a body written as an argument under `\ExplSyntaxOn`. Anything else is refused with an error. The interface is keyval because a mode may later have more than four things to set: new keys may appear in a future release, which is exactly what the arity-encoded form this replaced could not do without changing its own name.

`defaults` is required, and is what `\resetExdefaults` calls. It must set every length of *both* parameter sets described in section 8, since the document may switch modes afterwards. The three hooks are optional — one omitted does nothing, and the level keeps the list parameters it was handed. They run inside the list declaration of their level and may set `\labelwidth`, `\labelsep`, `\leftmargin` and `\topsep`; `main` runs after the label is fixed, so it may measure it (which is what `legacy` does). A hook that wants its level tagged with a numbering style calls `\lx_ol_class:n`. `\lx_mode_select:n` assigns locally, so a mode selected inside a group is undone at the end of it; the mode in force is readable in `\l_lx_mode_tl`.

12 Command reference

Command	Effect
Examples	
`\ex.`	Start a numbered example; a blank line ends it. `\ex.[`⟨*label*⟩`]` uses a custom label without advancing the counter.
`\a.`	Open a level of sub-examples (letters, then roman).
`\b. -- \f.`	Next item at the level currently open.
`\z.`	Leave one level; from the letter level or an example without sub-examples, end it.
`exe, xlist`	`gb4e`-compatible batch and sub-level environments (package option `gb4e`); items via `\ex`, `\ex[`⟨*judgment*⟩`]{`⟨*text*⟩`}`, or plain `\item`.
`\exg.`	`\ex.` followed by `\gll`. `\exg.[`⟨*label*⟩`]` takes a custom label too, written against the command: a bracket after a space is the object line's first word.
`\ag. -- \fg.`	`\a.`–`\f.` followed by `\gll`.
Judgments	
`\jdg{`⟨*mark*⟩`}`	Hang an arbitrary mark in the margin.
`\DeclareJudgment{`⟨*cmd*⟩`}{`⟨*mark*⟩`}`	Give a mark a name.
Glosses	
`\gll, \glll`	Two-, three-tier gloss; lines end in `\\`.
`\gl … \endgl`	Gloss with any number of tiers.
`\glt`	Free-translation line. (`\gln` is a synonym.)
`\CommonGloss`	A gloss line shared by the variant rows `\ag. … \fg.` above it, or by the rows of an `xlist[CommonGloss]` in an `exe` batch (section 5.12).
`\GlossTierFont{`⟨*n*⟩`}{`⟨*cmd*⟩`}`	Font command for tier ⟨*n*⟩.
`\GlossTierLang{`⟨*n*⟩`}{`⟨*code*⟩`}`	Language of tier ⟨*n*⟩ for tagged PDFs (section 5.9).
`\GlossTransStyle`	Declaration applied to the free translation after `\glt`; empty by default.
`\GlossTransLang{`⟨*code*⟩`}`	Language of the free translation for tagged PDFs; without it, the gloss's last tier's, if declared (section 5.9).
`\GlossTransSide`	Set the free translation in a column *beside* the gloss rather than under it (section 7.4); `\GlossTransBelow` restores the default. A declaration, and it must come before the example. Top-level examples only, and not on a gloss carrying an `\exannot`.

continued on the next page

Command	Effect
`\GlossTransRatio`	Share of the measure the gloss gets beside a side translation (`.6`, a factor, not a length).
`\GlossTransSep`	Gap between the two columns (`2em`).
`\GlossTransRightSkip`	Stretch at the right of the translation's column (`0pt plus 2em`); without it a narrow column overflows rather than ending short.
`\GlossTransMinWidth`	Below this the side position is abandoned and the translation set underneath, with a warning (`6em`).
`\lpzg{`*⟨abbr⟩*`}`	Gloss abbreviation in small caps, with spoken expansion when tagged (section 5.2).
`\SetLeipzig{`*⟨abbr⟩*`}{`*⟨expansion⟩*`}`	Add or override a Leipzig abbreviation expansion.
`\lpzglist[`*⟨keys⟩*`]`	List of the abbreviations the document uses, each with its full form (section 5.3). Keys: `style`, `sort`, `include`, `ignore`, `add`, `unexplained`, `title`, `titlestyle`, `sep`, `itemsep`, `format`.
`\lpzglistsetup{`*⟨keys⟩*`}`	The same keys, for every list.
`\lpzgcheck{`*⟨keys⟩*`}`	Consistency checks on the abbreviations: `unknown` (on by default), `unused`, `ignore` (section 5.4).
`\ag. … \fg.`	Glossed sub-example: `\a.` … `\f.` followed by `\gll`. A shorthand whose name another package already owns is left to it (section 10).
`\lpzgadd{`*⟨abbrs⟩*`}`	Register abbreviations used outside `\lpzg` so that they reach the list.
`\lpzglistentry{`*⟨abbr⟩*`}{`*⟨full form⟩*`}`	One entry of the list; redefine to restyle every list.
`\eachwordone/two/three`	Font commands for tiers 1–3.
`\GlossSep`	Glue between gloss columns.
`\GlossPhantomAlign`	Align gloss words past leading brackets, parentheses and judgment marks (*recommended*; section 5.7). Also the package option `phantomalign`; `\GlossPhantomAlignOff` reverts.
`\GlossPhantomChars{`*⟨marks⟩*`}`	Leading marks that alignment skips; one token is one mark, characters and commands alike (default `*?([<` plus \# \%).
`\GlossPhantom{`*⟨material⟩*`}`	Manual override: pad a gloss word by an invisible box the width of *⟨material⟩*, in the object font.
`\GlossMorphAlign`	Align morphemes inside each column, at `-` and `=` (off by default; section 5.8); `\GlossMorphAlignOff` reverts.
`\GlossRTL`	Set glosses as right-to-left grids, LuaLaTeX or XeLaTeX only (section 5.10); `\GlossRTLOff` reverts.

continued on the next page

Command	Effect
`\glscript{`*⟨lang⟩*`}{`*⟨text⟩*`}`	An unaligned line in the native script above a gloss, in its own direction and language (section 5.10).
References	
`\label, \ref`	As usual; references print in parentheses.
`\sublabel{`*⟨key⟩*`}`	Label a sub-example for use in a range.
`\Next, \NNext, \Last, \LLast`	Relative references; `[`*⟨letter⟩*`]` adds a sub-example part, as (3b). Clickable under hyperref.
`\TextNext`	Next example of the running text (from a footnote).
`\pref, \pNext, …`	Parenthesis-free twins of all of the above.
`\refrange{`*⟨a⟩*`}{`*⟨b⟩*`}`	Compact range, as (3a–c).
`\prefrange, \Refrange`	Without parentheses; over whole examples.
`\exrefcheck[`*⟨words=n⟩*`]`	List what each relative reference and range resolved to, with the target's opening words (section 6.5).
Further apparatus	
`\altn{`*⟨...⟩*`}{`*⟨...⟩*`}`	Stacked alternatives; `[`*⟨c/l/r⟩*`]` alignment. Fallback: `\lxAltn`.
`\altg{`*⟨alt⟩*`}{`*⟨alt⟩*`}...`	Glossed alternatives: written in both lines of \exg. (objects, then glosses); both-braced, centred. Also `\lxAltg`.
`\SetAltSpoken{`*⟨word⟩*`}`	The word spoken between stacked alternatives, for `\altn` and `\altg` alike: `\SetAltSpoken{`*⟨ou⟩*`}` gives "aa, bb ou cc". The starred form keeps the comma before the connector, which is English usage and the default; an optional `[`*⟨punct⟩*`]` replaces the punctuation. Respects grouping (section 9.2).
`\exsource{`*⟨text⟩*`}`	Flush-right attribution.
`\exannot[`*⟨spoken⟩*`]{`*⟨text⟩*`}`	Structural label, set in a column at `\ExAnnotColumn`; on the object line in a gloss. `\SetAnnotSpoken{`*⟨text⟩*`}{`*⟨phrase⟩*`}` registers a spoken form for a label used throughout (section 7.5).
`\ExAnnotFit`	Measure that column from the examples instead: one column per example, past the longest of them. Costs an `.aux` round trip; `\ExAnnotNoFit` turns it off again.
`\ExAnnotColumn`	Where the column is, from the *left* edge of the text block (`.75\columnwidth`).
`\ExAnnotSep`	Least gap between an example and that column (`1em`); give it no stretch or shrink.
`\ExAnnotFont`	Font of the annotation (`\normalfont`).

continued on the next page

Command	Effect
`\mvfrom[`⟨*options*⟩`]{`⟨*label*⟩`}{`⟨*text*⟩`}`, `\mvto[`⟨*options*⟩`]{`⟨*label*⟩`}{`⟨*text*⟩`}`	The base position and the landing site of a movement; one of each with the same label in one example draws an arrow to the `\mvto`. Options `above`, `below`, `level=`⟨*n*⟩, `spoken=`⟨*text*⟩. Two runs (section 7.6). Fallbacks: `\lxMoveFrom`, `\lxMoveTo`.
`\SetMoveSpoken{`⟨*word*⟩`}`	The note a screen reader says after a base position (`moved`). Respects grouping.
`\MoveArrowGap`, `\MoveArrowStep`	Gap between a word and its arrow (`.3ex`); distance between levels (`1.1ex`). Macros. The arrow's look is the TikZ style `lx move`.
Layout and options	
`lazy, gb4e`	Package options: dot syntax (default), `gb4e` environment syntax; combinable.
`phantomalign`	Package option: aligns words in glosses with the word in the object line, not a bracket or judgment mark; combinable.
`legacy`	Package option: the geometry and dash conventions of linguex; orthogonal to the syntax options.
`norelreflinks`	Package option: print \Next and \Last without the hyperref link.
`\resetExdefaults`	Restore all layout lengths to the current mode's defaults.
`\Exalph, \Exroman`	The sub-example letter and numeral, from a counter's name; `\alph` and `\roman` by default, so the language's. `\let\Exalph\ExLatinAlph` for Latin letters in any language.
`\firstrefdash`	Number–letter dash in "(3a)"; empty by default, `-` under `legacy`.
`\secondrefdash`	Letter–roman dash in "(3a-i)"; `-` in both modes.
`\rangedash`	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.