

ExPex

for linguists

Example formatting, glosses, and reference

- (1) a. *Mary_i ist sicher, [dass es den Hans nicht stören würde [seiner Freundin ihr_i Herz auszuschütten]].*
Mary is sure that it the-ACC Hans not annoy would his-DAT girlfriend-DAT her-ACC heart-ACC out to throw
'Mary is sure that it would not annoy John to reveal her heart to his girlfriend.'
- b. *Mary_i ist sicher, [dass seiner Freunden ihr_i Herz auszuschütten [dem Hans nicht schaden würde]].*
Mary is sure that his-DAT girlfriend-DAT her-ACC heart-ACC out to throw the-DAT Hans not damage would
'Mary is sure that to reveal her heart to his girlfriend would not damage John.'

User's Guide

John Frampton
j.frampton@neu.edu

January 2014
Version 5.0

Contents

| | | |
|----|---|----|
| 1 | Introduction | 1 |
| | 1.1 Changes | 1 |
| | 1.2 LaTeX/Tex cooperation | 2 |
| | 1.3 Acknowledgements | 3 |
| 2 | Some preliminary examples | 4 |
| 3 | XKV parameterization | 8 |
| 4 | Examples without parts | 10 |
| | 4.1 Explicit example numbers; Formatting the example number | 13 |
| 5 | Examples with labeled parts: Basics | 15 |
| | 5.1 nopreamble | 17 |
| | 5.2 Stipulated labels | 18 |
| 6 | More on examples, with and without labeled parts | 18 |
| | 6.1 Anchoring | 18 |
| | 6.2 Formatting the labels | 19 |
| | 6.3 Aligning the labels | 21 |
| | 6.4 Relative versus fixed dimensions | 23 |
| | 6.5 User designed labeling | 23 |
| | 6.6 The parameter <code>samplexno</code> | 25 |
| | 6.7 IJAL style format of multiline examples | 26 |
| | 6.8 Footnotes and endnotes | 28 |
| 7 | User defined styles | 30 |
| 8 | Judgment marks | 32 |
| 9 | Glosses | 34 |
| | 9.1 Parameters | 36 |
| | 9.2 Exceptional <code>\gla</code> items | 39 |
| 10 | Nlevel glosses; an alternate coding syntax | 42 |
| | 10.1 Parameters which modify particular lines | 44 |
| | 10.2 <code>\nogloss</code> and the diacritics <code>@</code> and <code>+</code> | 45 |
| | 10.3 Line spacing inside <code>glwords</code> | 45 |
| 11 | More about glosses | 46 |
| | 11.1 The parameter <code>glwidth</code> | 47 |
| | 11.2 Comments and citations | 48 |
| | 11.3 Line spacing in wrapped glosses | 49 |
| 12 | Glosses; Special topics | 51 |
| | 12.1 User defined levels | 51 |
| | 12.2 Positioning the free translation to the right of the interlinear gloss | 52 |

| | | |
|------|--|----|
| 12.3 | Glosses with a side panel | 53 |
| 12.4 | Cascading hanging indentation in glosses | 55 |
| 12.5 | Gloss underfixes | 56 |
| 12.6 | Center alignment in glwords | 58 |
| 13 | Referring to examples and labeled parts of examples | 60 |
| 13.1 | Unnamed reference | 60 |
| 13.2 | Named reference | 60 |
| 13.3 | Proofing references | 62 |
| 13.4 | The tag/reference file | 62 |
| 13.5 | References to references, references as values | 63 |
| 13.6 | Extensions of the tag/reference mechanism | 64 |
| 13.7 | The parameter <code>fullreformat</code> | 64 |
| 13.8 | Reference to a part of a multipart example | 65 |
| 13.9 | Support for the LaTeX <code>\label</code> and <code>\ref</code> commands | 66 |
| 14 | Tables in examples | 66 |
| 14.1 | Tables with labeled lines | 68 |
| 14.2 | Tagging implicit labels in tables | 69 |
| 14.3 | Some useful table making tools | 70 |
| 14.4 | Tables that can break between pages | 71 |
| 14.5 | Squeezing tables into tight places | 73 |
| 15 | ExPex and PSTricks | 74 |
| 16 | Control over page breaking inside examples | 75 |
| 16.1 | Discouraging page breaks in examples | 76 |
| 16.2 | Controlling where page breaks occur in examples (if they must) | 77 |
| | Index of control sequences, parameters, and special symbols | 78 |

1. Introduction

Many of the needs of linguists who wish to produce typographically attractive papers using *TeX* or *LaTeX* are not specific to linguistic papers. There are therefore many macro packages which deal with tables of contents, references, section headings, font selection, indexing, etc. But linguistics does have some special typographic needs. I addressed two of these with the macro packages *PST-JTree* and *PST-ASR*, which typeset syntactic trees and autosegmental representations. *ExpPex* addresses the main remaining special *TeX* need in linguistics: formatting examples, examples with multiple parts, glosses, and the like, and referring to examples and parts of examples. The name comes from the two central macros, `\ex` and `\pex`, used to typeset examples and examples with labeled parts.

PST-JTree and *PST-ASR* rely heavily on Hendri Adriaens' *XKeyVal* package, which has become the standard for *PSTricks* based macro packages. Although *ExpPex* is not based on *PSTricks*, it does handle parameterization the same way that *PST-JTree* and *PST-ASR* do. When *expex.tex* is loaded, it immediately checks to see whether *xkeyval.tex* has already been loaded. If not, it does so.

The goal in writing a macro package for general use is to make it simple to use if only simple things need to be done, but powerful enough so that users who have complex needs can get those needs satisfied if they are willing to deal with the complexities that complex needs inevitably involve. If you think there are simple things that are not simple to do, or complex things that cannot be done, please write to me at j.frampton@neu.edu. The *ExpPex* macros have evolved over the last 15 or so years and like anything which evolves, various features of the current state may have more to do with history than with optimal design. Please let me know about departures from optimal design. Perhaps the next version can be improved.

This User's Guide begins with four examples which demonstrate *ExpPex* in action. It serves as a "A Quick Guide to *Expex*". Each page gives some code at the top, with the product of this code below. Its main purpose is to give a sense of how *ExpPex* works, so that curious readers have some basis for determining whether they want to proceed with the details. It is possible to begin to use *ExpPex* solely on the basis of the four demo pages and learn the more subtle capabilities as needed. A quick survey of the index and the table of contents should give you some idea of what is available, if you need it.

1.1. Changes

The most recent previous version of *ExpPex* is version 4.1d. That version will still be available on CTAN, zipped as *expex41d.zip*. Aside from bug fixes, the changes in this version are:

1. The gloss parameter `abovemoreglskip` has been deprecated (declared obsolete and slated to be eliminated) for some time. It has now been removed from *ExpPex*. The function it served is now served in a more general way by `extraglskip`.
2. An entirely new way of coding glosses, called the *nlevel style* has been introduced. See Section 10 for the many details. This style makes it much easier to keep track of words and their associated glosses, particularly when long glosses need to be typeset.
3. A new way is introduced for including un glossed items (brackets, for instance) among the items which are glossed. The exceptional gloss items '[' and ']' are now deprecated as exceptional gloss items. Consequentially, the parameters `everybrack`, `glbrackwordsep`, and `glbrackbracksep` have also been deprecated. All these deprecated items will be removed from *ExpPex* at some time in the future.

4. Control over vertical spacing in glosses has been streamlined. Two little used parameters `autoglskip` (which was rarely used outside the default setting) and `gllineskip` have been eliminated. Their functions have been taken over by `extraglskip` and strut insertion. It is possible that this may create some problems with backwards compatibility. If there are problems, either use Version 4.1d or contact me directly to see if there is an easy solution to the problem.
5. Glosses now break over pages, provided they are full width.
6. Center alignment is now an option for glosses. Glosses of words can be centered under the word they gloss.
7. *ExPex* example numbering is now compatible with the *LaTeX* `\includeonly` mechanism.

1.2. LaTeX/TeX cooperation

ExPex is designed to be used by either *Plain Tex* or *LaTeX* users. *LaTeX* users need to say `\usepackage{expex}` and *Tex* users `\input expex`. All of the code for the examples in this documentation should run equally well in either system, subject to the notes below.

1.2.1 Note to LaTeX users

`\it` (now a deprecated *LaTeX* command) is used in a few places. In case *ExPex* detects that *LaTeX* is being used, it executes `\let\it=\itshape`.

1.2.2 Note to TeX users

Three macros are used in the examples in this documentation which are defined in *LaTeX* but not in *Plain Tex*: `\footnotesize`, `\sc`, and `\textsc`. Assuming, for example, that text is set in 10pt computer modern, the following would suffice for all the examples in this documentation.

```
\font\eightrm=cmr8
\font\eightit=cmti8
\def\footnotesize{\eightrm \let\it=\eightit \baselineskip=9pt}
\font\tensc=cmcsc10
\let\sc=\tensc
\def\textsc#1{{\sc #1}}
```

Most *Plain Tex* users will have other fonts and much more general size changing macros at their disposal. The code above is barely sufficient to handle the examples in this is documentation, but it will do the job. For what it is worth, this documentation was typeset using *Plain Tex*. `\twelvepoint` and `\tenpoint` were defined modeled on pages 414–415 in the *TeXbook* and `\let\footnotesize=\tenpoint` was executed to define `\footnotesize`. The running text is 12pt. `\sc` was defined by `\font\twelvesc=cmcsc10 scaled\magstep1`¹ and `\let\sc=\twelvesc`².

1. `cmcsc10 scaled` was used rather than `cmcsc12` because Postscript fonts for `cmcsc10` are more readily available than those for `cmcsc12`.

2. Small caps are used only in text size

1.3. Acknowledgements

Many participants in the *Ling-Tex* discussion group have contributed to the development of *ExPex*, either by posing good questions, solving problems, or providing informed discussion of desirable features. In particular, I thank Stephen Anderson, Mario Bisiada, Noah Constant, James Crippen, Alexis Dimitriadis, Claude Dionne, Kevin Donnelly, Antonio Fortin, Jeremy Hammond, Daniel Harbor, Joshua Jensen, Don Killian, Joost Kremers, John Lyon, Alan Munn, Christos Vlachos, and Natalie Weber.

2. Some preliminary examples

1. *Examples and examples with parts*

Example (\nextx) is well-known from the literature on parasitic gaps. Here we are concerned with example formatting, not with the interesting syntax.

```
\ex
I wonder which article John filed {\sl t\} without reading {\sl e}.
\xe
It is beyond the scope of this investigation to determine exactly why
John did not read the article.
```

Multipart examples are equally straightforward.

```
\pex Two examples of parasitic gaps.
\ax He is the man that John did not interview {\sl e\} before
he gave the job to {\sl e}.
\ax He is someone who John expected {\sl e\} to be successful
though believing {\sl e\} to be incompetent.
\xe
Here, we can speculate on why John did not do an interview before
recommending the person for a job. It is likely that the person
was a crony of John. In (\lastx b), perhaps John knew that
the “someone” went to prep school with the owner of the
business.
```

Example (2) is well-known from the literature on parasitic gaps. Here we are concerned with example formatting, not with the interesting syntax.

(2) I wonder which article John filed *t* without reading *e*.

It is beyond the scope of this investigation to determine exactly why John did not read the article.
Multipart examples are equally straightforward.

(3) Two examples of parasitic gaps.

a. He is the man that John did not interview *e* before he gave the job to *e*.

b. He is someone who John expected *e* to be successful though believing *e* to be incompetent.

Here, we can speculate on why John did not do an interview before recommending the person for a job. It is likely that the person was a crony of John. In (3b), perhaps John knew that the “someone” went to prep school with the owner of the business.

2. Named reference

If examples and parts of examples are tagged, they can be referred to by name.

```
\pex<pg>
\A This is the man that John interviewed {\sl e\} before
telling you that you should give the job to~{\sl e}.
\A<A> This is someone who John expected {\sl e\} to be successful
though believing {\sl e\} to be incompetent.
\Xe
```

Now, names can be used. The name/reference pairs can be written to a file, making forward reference possible and backwards reference at a distance reliable. You can refer to part `\getref{pg.A}` of example (`\getref{pg}`), or (`\getfullref{pg.A}`).

If you use a tag that has not been defined, `{\sl ExPex\}` will let you know. If you try to reference a name which has no reference, `\getref{pg.B}` for example, a warning will be issued and the (bracketed) tag printed as shown at the beginning of this sentence. If you try to tag a part of an example which has no tag, `{\sl ExPex\}` will let you know about that as well.

If examples and parts of examples are tagged, they can be referred to by name.

- (4) a. This is the man that John interviewed *e* before telling you that you should give the job to *e*.
- b. This is someone who John expected *e* to be successful though believing *e* to be incompetent.

Now, names can be used. The name/reference pairs can be written to a file, making forward reference possible and backwards reference at a distance reliable. You can refer to part b of example (4), or (4b).

If you use a tag that has not been defined, *ExPex* will let you know. If you try to reference a name which has no reference, [pg.B] for example, a warning will be issued and the (bracketed) tag printed as shown at the beginning of this sentence. If you try to tag a part of an example which has no tag, *ExPex* will let you know about that as well.

3. Glosses

```

\ex
\beginl
\gla Mary$_i$ ist sicher, dass es den Hans nicht st\oren
w\urde seiner Freundin ihr$_i$ Herz auszusch\utten.//
\glb Mary is sure that it the-{\sc acc} Hans not annoy would
his-{\sc dat} girlfriend-{\sc dat} her-{\sc acc} heart-{\sc acc} {out to
throw}//
\glft ‘Mary is sure that it would not annoy John to reveal her
heart to his girlfriend.’//
\endgl
\xe

```

(5) *Mary_i ist sicher, dass es den Hans nicht stören würde seiner Freundin ihr_i Herz auszuschütten.*
 Mary is sure that it the-ACC Hans not annoy would his-DAT girlfriend-DAT her-ACC
 heart-ACC out to throw
 ‘Mary is sure that it would not annoy John to reveal her heart to his girlfriend.’

Glossing can be done in two different coding styles. Either of the following produce the gloss display below.

| | |
|---|--|
| <pre> \ex \beginl \gla k- wapm -a -s'i -m -wapunin -uk// \glb Cl V Agr Neg Agr Tns Agr// \glc 2 see {\sc 3acc} {} {\sc 2pl} preterit {\sc 3pl}// \glft ‘you (pl) didn’t see them’// \endgl \xe </pre> | <pre> \ex \beginl[glstyle=nlevel] k-[Cl/2] wapm[V/see] -a[Agr/\sc 3acc] -s'i[Neg] -m[Agr/\sc 2pl] -wapunin[Tns/preterit] -uk[Agr/\sc 3pl] \glft ‘you (pl) didn’t see them’ \endgl \xe </pre> |
|---|--|

(6) *k- wapm -a -s'i -m -wapunin -uk*
 Cl V Agr Neg Agr Tns Agr
 2 see 3ACC 2PL preterit 3PL
 ‘you (pl) didn’t see them’

3. XKV parameterization

Macro: `\lingset`

(Here and in following sections and subsections, an inventory of the macros, parameters, and count registers which are described in what follows appears at the beginning of the section. “What follows” refers to the text up to the next section or subsection heading.)

The key-value approach to parameter setting in *TeX*, which originated with David Carlisle’s `keyval` package, is illustrated by the key `textoffset`. *ExPex* makes the distance from the example number to the text equal to value associated with this key. Executing the command

```
\lingset{textoffset=1.3em}
```

results in the definition (or redefinition) of the macro `\lingtextoffset` so that it expands to the value 1.3em. The macro `\ex`, which is used to typeset examples without labeled parts, uses `\lingtextoffset`. But *ExPex* users never have to concern themselves with the macro `\lingtextoffset`. If they are not satisfied with the default spacing, they simply have to know that `textoffset` is the key for setting the distance from example number to the text.

The argument of `\lingset` can be a comma separated sequence of key/value pairs. The syntax is:

```
\lingset{key1=value1, ..., keyn=valuen}
```

The comma separated key/value pairs are processed sequentially, from left to right. If a value contains a comma, it must be hidden from the mechanism which parses the list by putting the value in braces. The braces are removed by the parser.

Many *ExPex* macros take an optional argument, delimited by brackets, which is passed to `\lingset`. `\ex`, for example, takes an optional argument. You might say, for example,

```
\ex[textoffset=1.4em,aboveexskip=0pt]
```

The argument of `\ex` will be passed to `\lingset` and the result evaluated, so that the example will be typeset with these parameter settings. This is carried out inside a group, so the global settings of the parameters are not affected. As we will see later, `aboveexskip=0pt` will cause the example to be typeset with no vertical skip above it. This is sometimes useful in avoiding exaggerated spacing when an example directly follows another one, with no intervening text.

ExPex has various kinds of keys. The distinctions depend on the effect of executing (8) and the restrictions on the possible values which can appear.

(8) `\lingset{key=value}`

Command key: After (8) is executed, the macro `\lingkey` or `\ling@key` expands to *value*. It will be made clear when the key is introduced whether it is `\lingkey` or `\ling@key` that is defined.³

Incremental dimension parameter: *value* must be a dimension or a dimension prefixed by `!`. If *value* is a dimension, it is stored in `\lingkey`. If it is a `!`-prefixed dimension, the prefixed dimension is added to its former dimension and the result stored in `\lingkey`. There must be

3. If there was a significant chance that some users might want easy access to the macro value, `\lingkey`, with no `@` in the macro name, was used.

a dimension to increment, so a fatal error results if `\lingkey` does not expand to a dimension. Incremental parameters are very useful if a minor adjustment to the format in a particular example is desired. There are *incremental skip parameters* as well, which operate in an entirely parallel manner.

In the case of command and incremental keys, the notation `Val(key)` is used to indicate the value of the key. So, for example, `Val(textoffset)` is the value associated with the key `textoffset`.

Pseudo parameter: (8) is executed for its side effects. A *value* is not stored. The key `samplelabel` illustrates this. When `\lingset{samplelabel=A.}` is executed, no *ExPex* macro which expands to *value* is defined. Instead, the parameter `labelwidth` is set to the width of “A.” in the current font.

Choice parameter: Choice parameters are a kind of pseudo parameter. The *value* which is assigned must be drawn from a prescribed list. *ExPex* choice parameters do not store *value* when (8) is executed; the purpose of executing (8) is the side effects which are coded into the definition of the key.

The parameter `labelalign` illustrates this. `\lingset{labelalign=value}` is valid only if *value* is one of `left`, `center`, or `right`. A fatal error results otherwise. The effect is to appropriately define the macro `\@labelprint` which is used to typeset the labels of subparts in multipart examples.

When XKV keys are defined, they can be given default values, as part of their definitions. If a key `foo`, for example, is given the default value `2pt`, then executing `\lingset{foo}` is equivalent to executing `\lingset{foo=2pt}`. Only a few *ExPex* keys have default values in the XKV sense, but most are set to an initial value in *expex.tex*. This initial value will sometimes be called the default setting of the key, even though the key does not have a default setting in the XKV sense.

4. Examples without parts

Sections will usually begin with an inventory of the parameters and user friendly macros and registers that are introduced in the section, The initial settings of parameters will be given if they are relevant. Pseudo parameters are set for their immediate effect, not to store a setting associated with the parameter, so the initial setting of pseudo parameters is not relevant. `\ex~[]` above should be taken to mean that the macro `\ex` will be described, that it is optionally modified by a following diacritical tilde `~` (as described below), and that it optionally takes an argument. The text will describe what arguments are permitted. A \dagger superscript on a parameter *key* in these inventories indicates a parameter whose value is accessible by the macro `\lingkey`. A (!) prefix on “dimension” or “skip” indicates that the parameter is incrementable, by a dimension or a skip as the case may be.

Macros: `\ex~[]`, `\xe`

Counter: `\excnt`

Parameters:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|--|--------------|----------------------------|
| <code>numoffset\dagger</code> | (!)dimension | 0pt |
| <code>textoffset\dagger</code> | (!)dimension | 1em |
| <code>aboveexskip\dagger</code> | (!)skip | 2.7ex plus .4ex minus .4ex |
| <code>belowexskip\dagger</code> | (!)skip | 2.7ex plus .4ex minus .4ex |
| <code>exskip</code> | skip | (pseudo-parameter) |

`\ex` constructions are terminated by `\xe`. The following sample paragraph illustrates the use of `\ex ... \xe`. The convention in this manual is that text, *as it would appear in a document*, is displayed in a framed box, usually with the code immediately following or preceding. The code assumes that the initial parameter settings are in effect at the point that the code is executed.

The following example is well-known from the literature on parasitic gaps. Here we are concerned with example formatting, not with the interesting syntax.

(9) I wonder which article John filed *t* without reading *e*.

Various aspects of the format are controlled by parameters, which can be set either globally or via an optional argument.

The following example is well-known from the literature on parasitic gaps. Here we are concerned with example formatting, not with the interesting syntax.

```
\ex
I wonder which article John filed {\sl t\} without reading {\sl e}.
\xe
```

`\noindent` Various aspects of the format are controlled by parameters, which can be set either globally or via an optional argument.

Those users who try to save virtual paper can equally use:

The following example is well-known from the literature on parasitic gaps. Here we are concerned with example formatting, not with the interesting syntax.`\ex` I wonder which article John filed `{\sl t\}` without reading `{\sl e}`.`\xe` Various aspects of the format are controlled by parameters, which can be set either globally or via an optional argument.

With different parameter settings, we get:

The following example is well-known from the literature on parasitic gaps. Here we are concerned with example formatting, not with the interesting syntax.

(11) I wonder which article John filed *t* without reading *e*.

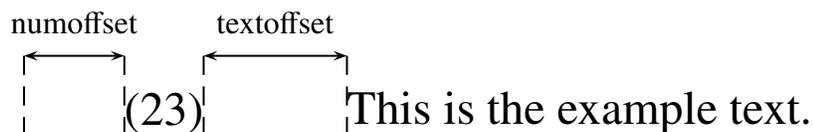
Various aspects of the format are controlled by parameters, which can be set either globally or via an optional argument.

The following example is well-known from the literature on parasitic gaps. Here we are concerned with example formatting, not with the interesting syntax.

```
\ex[numoffset=2em,textoffset=.5em,aboveexskip=1ex,belowexskip=1ex]
I wonder which article John filed {\sl t\} without reading {\sl e}.
\xe
```

`\noindent` Various aspects of the format are controlled by parameters, which can be set either globally or via an optional argument.

The horizontal dimensions are illustrated below. `numoffset` is measured from the left margin.



Example numbering is automatic. The count is kept in `\excnt`. It is incremented when `\ex` is expanded, before the number is typeset. `\excnt` will therefore always give the count of the next example. Note that this means that inside `\ex... \xe`, `\excnt` does not give the count of the current example. Vertical skip is inserted before and after examples, of amounts determined by `Val(aboveexskip)` and `Val(belowexskip)`.

Inside `\ex` constructions, the example text is wrapped as ordinary text, with `\leftskip` set by `\ex`. Since `\ex` sets `\leftskip` and relies on this setting, changes in `\leftskip` inside `\ex ... \xe` must be made with care, but can be made after the first paragraph (i.e. after the first explicit or implicit `\par`).

- (13) Und hier können wir sehen was für Unfug wird gemacht wenn er einen ganz langen Satz binnen kriegt.
- (14) α governs β if $\alpha = X^0$ (in the sense of X-bar theory), α c-commands β , and β is not protected by a maximal projection.

The code which was used to typeset the pair of examples above has two useful features which are worth highlighting.

```
\ex
Und hier können wir sehen was für Unfug wird gemacht
wenn er einen ganz langen Satz binnen kriegt.\par\nobreak
\xe
```

```
\ex[aboveexskip=0pt]
 $\alpha$  {\it governs\}/}  $\beta$  if  $\alpha = X^0$  (in the
sense of X-bar theory),  $\alpha$  c-commands  $\beta$ , and  $\beta$ 
is not protected by a maximal projection.
\xe
```

`\par\nobreak` is used to illustrate how a page break between two consecutive examples can be suppressed. This is sometimes desirable. `\par` puts *TeX* in the mode of adding lines to the page, and `\nobreak` tells *TeX* to avoid a break (which is a page break when *TeX* is in the mode of adding lines), essentially until after more text is added to the page. `aboveexskip=0pt` is used in the second example to avoid double spacing between the examples. Otherwise vertical skip would be added both below the first example and above the second example.

Since the need to suppress vertical skip above examples arises with some frequency, a shortcut is made available to accomplish this. Simply say `\ex~`. Tilde modification of `\ex` can be used with parameters; `\ex~ [...]` will be interpreted correctly.

`exskip` is a pseudo parameter which can be used to simultaneously set both `aboveexskip` and `belowexskip`. The effect of `\lingset{exskip=value}` is

```
\lingset{aboveexskip=value,belowexskip=value}
```

4.1. Explicit example numbers; Formatting the example number

Parameters:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|------------|----------------------------------|----------------------|
| exno | token list | (pseudo parameter) |
| exnoformat | token list of the form ... X ... | (X) |
| exnotype | arabic, chapter.arabic, or roman | arabic |

4.1.1 exno and exnoformat

Suppose that you want to repeat an example that was given earlier in your document. Something like.

(15) This is a crucial example.

It is clear that this example is related to the earlier example (5), which is repeated below.

(5) This is an example that was given many pages earlier.

If we are on the right track, as the saying goes, we expect the next example to be grammatical. But it is not.

(16) * ...

```
\ex This is a crucial example.\xe
```

```
It is clear that this example is related to the earlier  
example (14), which is repeated below.
```

```
\ex[exno=14]
```

```
This is an example that was given many pages earlier.\xe
```

```
If we are on the right track, as the saying goes,  
we expect the next example to be grammatical. But it is not.
```

```
\ex * \dots\xe
```

excnt is not incremented if the example number is supplied by exno.

exno does not have to be set to an integer, as shown below.

```
\ex[exno=$\Delta$] Earlier example.\xe
```

(Δ) Earlier example.

Sometimes, it is desirable to have an alternative to the standard formatting of example numbers with parentheses.

```
\ex[exno={14, repeated},exnoformat=X] Earlier example.\xe
```

| |
|---------------------------------|
| [14, repeated] Earlier example. |
|---------------------------------|

Note the use of braces to hide the comma in the setting of `exno`. Otherwise, the key-value parser would get confused, interpreting `[14` as the setting of `exno` and reporting that `repeated` is an undefined key. The initial setting of `exnoformat` is `(X)`, so this parameter must be reset to prevent putting parentheses around the special `exno`. The label formatting mechanism is primitive. `labelformat` must be of the form

⟨balanced text⟩X⟨balanced text⟩

The pre-`X` text is inserted before the label (including the material specified by `everypar`) and the post-`X` text is inserted after the label. The balanced text cannot contain the character `X`. *Balanced text* is a string of tokens with properly nested (explicit) braces. No error checking is done to ensure that the format specification has the required form, so be careful. An error might lead to very obscure error messages.

Note that

```
\ex[exno={14, repeated},exnoformat={X}] Earlier example.\xe
```

and

```
\ex[exno={14, repeated},exnoformat=X] Earlier example.\xe
```

give the same result. Note also that the brackets in key values are hidden inside braces.

Section (13.2) will show how to name example numbers, so that `exno` can be set by giving the name of an example number.

4.1.2 `exnotype`

In articles it is usual to number examples using consecutive arabic numerals. Setting `exnotype` to `arabic` accomplishes this. Footnotes generally use consecutive roman numerals (i), (ii), etc., requiring `exnotype` to be set to `roman`. See Section 6.8 for numbering in footnotes.

Some books number examples using consecutive arabic numerals, restarted at 1 in each chapter, with the chapter number as a prefix. So (5.6), for example, would be the (6) example in chapter 5. Setting `exnotype` to `chapter.arabic` accomplishes this, provided that the `\chapter` macro which is employed cooperates and that the macro `\thechapter` produces the chapter number. Minor surgery on the `\chapter` macro in *LaTeX* classes which provide chapters is necessary. It must be modified so that `\excnt` is reset to 1 at the start of each chapter. The file `epltxchapno.sty` which is provided in this distribution provides a modification which works with `book.cls`, `scrbook.cls`, `report.cls`, and `memoir.cls`.

5. Examples with labeled parts: Basics

Counter: `\pexcnt`

Macros: `\pex []`, `\a[]`

Parameters:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|---------------------------------|--------------------------|---------------------------------------|
| <code>labeltype</code> | name of a key-value list | <code>alpha</code> |
| <code>labeloffset†</code> | (!)dimension | <code>1em</code> |
| <code>labelwidth†</code> | (!)dimension | <code>.72em</code> |
| <code>preambleoffset†</code> | (!)dimension | <code>1em</code> |
| <code>interpartskip†</code> | (!)skip | <code>1ex plus .2ex minus .2ex</code> |
| <code>belowpreambleskip†</code> | (!)skip | <code>1ex plus .2ex minus .2ex</code> |
| <code>samplelabel</code> | token list | (pseudo-parameter) |

Typical examples are given below, with the initial parameter settings.

- (17) a. This is the first example.
 b. This is the second example.
- (18) Multipart examples often have a title or preamble of some kind.
 a. This is the first example.
 b. This is the second example.

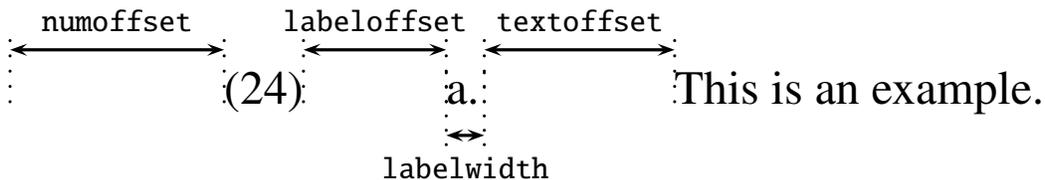
```
\pex
\a This is the first example.
\a This is the second example.
\xe
```

```
\pex~ Multipart examples often have a title or preamble of some kind.
\a This is the first example.
\a This is the second example.
\xe
```

Just like `\ex`, `\pex` must be closed by `\xe`, can be modified by a tilde diacritic to suppress adding vertical space above the example, and accepts parameters. The macro `\a`, which introduces each labeled part, is defined only within `\pex ... \xe`. It accepts certain parameters. Extra vertical skip (set by `interpartskip`) is inserted between the parts; and extra vertical skip (determined by `belowpreambleskip`) is inserted between the *preamble* and the first part. The preamble is the visible material, if any, that appears after the example number and before the first part.

The horizontal dimensions are parameterized as pictured below, provided that the anchoring parameters have their initial values. The parameters `numoffset` and `textoffset` are used in

both `\ex` and `\pex` constructions. The effects of changing the settings of the anchoring parameters (`labelanchor` and `textanchor`) will be considered in Section 6.1.



Adjustment for the width of the example number is automatic, but the width of the label slot is a parameter setting, not adjusted to the width of the particular label which appears in the label slot. The initial setting of `labelwidth` is the width of “a.” at the point that the default setting is established. This does not automatically change if the font is changed, in a footnote for example. It can be set by the user explicitly by setting `labelwidth` to the desired dimension, or as an incremental change to its previous value (`labelwidth=!3pt`, for example, increases the width of the label slot by 3pt). It can also be set indirectly by giving a sample label. `labelwidth` is then set to the current width of that sample. *ExPex* sets the default label width by `samplelabel=a..`

ExPex comes with three label types predefined: alpha, caps, numeric, and roman.

```
\pex[labeltype=alpha]
\a First part.
\a Second part.
\xe
```

| |
|--|
| (19) a. First part. b. Second part. |
|--|

```
\pex[labeltype=caps]
\a First part.
\a Second part.
\xe
```

| |
|--|
| (20) A. First part. B. Second part. |
|--|

```
\pex[labeltype=numeric]
\a First part.
\a Second part.
\xe
```

| |
|--|
| (21) 1. First part. 2. Second part. |
|--|

```
\pex[labeltype=roman]
\a First part.
\a Second part.
\xe
```

| |
|---|
| (22) (i) First part. (ii) Second part. |
|---|

This kind of labeling is common in footnotes.

Section 6 will detail all the parameters relevant to the label types and how additional label types can be defined by the user.

5.1. nopreamble

Parameter:

| <i>key</i> | <i>value</i> | <i>initial value</i> | <i>default value</i> |
|-------------------------|--------------|----------------------|----------------------|
| <code>nopreamble</code> | boolean | (not relevant) | true |

In order to properly format examples with parts, there are various reasons that `\pex` must be able to figure out whether or not there is a preamble. One reason is fairly obvious. If there is a preamble, every part introduced by `\a` must start on a new line, otherwise only parts after the first part start a new line. `\pex` tries to figure it out without help. If `\a` directly follows `\pex(^[...])` and perhaps a following space, `\pex` knows there is no preamble and acts accordingly. There are a few other following tokens, considered later in this manual, that `\pex` also knows are not signals that there is a preamble. But `\pex`'s preamble detection abilities are primitive. The following, for example, will confuse `\pex`.

```
\pex \it
\a first
\a second
\xe
```

(23)

a. first
b. second

The `\it` command is interpreted as a preamble.

`\pex` needs some guidance in this case. It is provided by the parameter `nopreamble`.

```
\pex[nopreamble=true] \it
\a first
\a second
\xe
```

(24) *a. first*

b. second

In fact, it is a little simpler than this. XKV parametrization allows one to stipulate a default value for each key that is defined. If the key is given to the parameter setting machinery with no value, then the key is set to the default value. `nopreamble` has the default value `true`. So the following is sufficient.

```
\pex[nopreamble] \it
\a first
\a second
\xe
```

(25) *a. first*

b. second

The global setting of `nopreamble` is irrelevant to `\pex`, which always assumes there is a preamble unless it sees an immediately following `\a` or `\pex` or is told directly that there is no preamble. In the following, for example, the setting of `nopreamble` outside `\pex` has no effect.

```
\lingset{nopreamble=true}
\pex \it
\a first
\a second
\xe
```

(26)

a. first
b. second

5.2. Stipulated labels

Parameter:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|------------|--------------|----------------------|
| label | token list | {} |

label is recognized as a key only by the `\a` macro. If `label=value` is passed to `\a`, that value is inserted as the label, ignoring automatic label generation.

```
\pex[exno=47]
\a[label=b]
\a[label=d]
\a[label=g]
\xe
```

| |
|---------------------|
| (47) b. d. g. |
|---------------------|

Besides `label`, the only other key that `\a` recognizes is `tag`. See Section 13.2.

6. More on examples, with and without labeled parts

In this section, we take up complications and fine points. Users should ignore it until they face a problem that the earlier sections do not deal with.

6.1. Anchoring

Parameters:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|----------------|------------------------------|----------------------|
| labelanchor | numright, numleft, or margin | numright |
| preambleanchor | numright, labelleft, or text | numright |
| textanchor | numleft or normal | normal |

Initially, the left edge of the label slot is determined by its offset from the right edge of the number, the left edge of the preamble is also determined by its offset from the left edge of the number, and the left edge of the text is determined by its offset from the right edge of the label slot. But the “anchor” for an offset is parametrized. In the example below, it is useful to anchor the `labeloffset` at the left edge of the example number.

```
\pex[exno={ [47, partially repeated from
p. 32] },labelanchor=numleft,
exnoformat=X,labeloffset=1.5em]
\par
\a[label=b] first
\a[label=d] second
\a[label=g] third
\xe
```

| |
|--|
| [47, partially repeated from p. 32] b. first d. second g. third |
|--|

Braces are needed in the specification of the value of `exno`, otherwise `\pex` would think that the key `exno` was set to the value [47 and try to set the key partially repeated from p. 32 to its default value. `exnoformat` is set to `X` so that parentheses are not put around the special `exno`.

The following would work equally well:

```
\pex[exno={47, partially repeated from
  p. 32},labelanchor=numleft,exnoformat={ [X] },
  labeloffset=1.5em]
  :
```

Braces are needed around `[X]` so that the mechanism that reads the optional argument of `\pex` does not interpret the right bracket as the right delimiter of the optional argument.

The normal setting of `textanchor` anchors the text at the right edge of the number for `\ex` constructions and the right edge of the label slot for `\pex` constructions.

Here is another example:

```
\lingset{textanchor=numleft,
  labelanchor=numleft,
  labeloffset=.35in,
  textoffset=.7in}
\pex[exno=9]
\ a first
\ a second
\ xe
\bigskip
\pex[exno=10]
\ a first
\ a[label=aa] second
\ xe
```

| | | |
|------|-----|--------|
| (9) | a. | first |
| | b. | second |
| (10) | a. | first |
| | aa. | second |

Some publications demand this style, in which both the label and text offsets are measured from the left edge of the number, or from the margin. It is a relic of typewriter days with mechanical tabs.

6.2. Formatting the labels

Parameters:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|--------------------------|------------------------|----------------------|
| <code>everylabel</code> | token list | <code>{}</code> |
| <code>labelformat</code> | <code>... A ...</code> | <code>A.</code> |

The token list `everylabel` is inserted just before labels are typeset. It is grouped so that it affects only the label. The main use is to set the font used for the labels if it differs from the font in the running text. For example:

```
\pex[everylabel=\it]
\ a one
\ a two
\ xe
```

```
(27) a. one
      b. two
```

There are other uses aside from setting the label font.

```
\pex[everylabel=A,labeltype=numeric,
      samplelabel=A1.]
\ a An example
\ a An example
\ a An example
\ xe
```

```
(28) A1. An example
      A2. An example
      A3. An example
```

The effect of the value of `labelformat` is illustrated in (29).

```
\pex[labelformat=$\langle A \rangle$,
      samplelabel=$\langle a \rangle$]
\ a first
\ a second
\ xe
```

```
(29) ⟨a⟩ first
      ⟨b⟩ second
```

The example above is fanciful, but one sometimes sees examples in the format below.

```
\pex[exnoformat=X.,labeltype=roman,
      labelformat=(A),samplelabel=(iii)]
\ a first
\ a second
\ a third
\ a fourth
\ xe
```

```
30. (i) first
     (ii) second
     (iii) third
     (iv) fourth
```

Like the example number formatting mechanism, the label formatting mechanism is primitive. `labelformat` must be of the form

⟨balanced text⟩ A ⟨balanced text⟩

The pre-A text is inserted before the label (including the material specified by `everylabel`) and the post-A text is inserted after the label. The balanced text cannot contain the character A. *Balanced text* is a string of tokens with properly nested (explicit) braces. No error checking is done to ensure that the format specification has the required form, so be careful. An error might lead to very obscure error messages.

There is some redundancy. The following is an alternate way to get the effect in (27) using `labelformat` instead of `everylabel`.

```
\pex[labelformat=\it A.]
\ a one
\ a two
\ xe
```

```
(31) a. one
      b. two
```

Another style sometimes found in footnotes is like the one in (30), except that the labels are right aligned in the label slot.

6.3. Aligning the labels

Parameters:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|-------------------------|------------------------|----------------------|
| <code>labelalign</code> | left, right, or margin | left |

There is a choice of left, right, or center alignment of the labels in the label slot. This is chosen by the parameter `labelalign`, which can be set to `left`, `center`, or `right`. For example,

```
\pex[exno=43,labeltype=roman,
  labelformat=(A),labelalign=right,
  samplelabel=(iii)]
\ a first
\ a second
\ a third
\ a fourth
\ xe
```

| |
|----------------|
| (43) (i) first |
| (ii) second |
| (iii) third |
| (iv) fourth |

This style looks odd to me, but this is the style for multipart examples *in the main text* in Chomsky's *Lectures on Government and Binding*. In footnotes, the style is:

```
\pex[exno=i,labeltype=alpha,
  samplelabel=(a),labelformat=(A)]
\ a first
\ a second
\ a third
\ a fourth
\ xe
```

| |
|---------------|
| (i) (a) first |
| (b) second |
| (c) third |
| (d) fourth |

If you look carefully, the vertical column of parts labels in the last example looks somewhat ragged because the width of “(b)” is slightly larger than the width of “(c)”. Center alignment of the labels gives a neater appearance.

```
\pex[exno=i,labeltype=alpha,
  samplelabel=(a),labelformat=(A),
  labelalign=center]
\ a first
\ a second
\ a third
\ a fourth
\ xe
```

| |
|---------------|
| (i) (a) first |
| (b) second |
| (c) third |
| (d) fourth |

For more ordinary `\pex` constructions which use the letters or numbers which have roughly the same width, label alignment is not a significant concern. But if labels include, for example, the narrow letter “i” and the wide letter “m”, as below, label alignment has a noticeable effect on the appearance. Individual tastes (and publisher's demands) may differ, but I prefer center alignment in these cases.

- | | | |
|---------------------------------|-----------------------------------|----------------------------------|
| (32) <i>left aligned labels</i> | (33) <i>center aligned labels</i> | (34) <i>right aligned labels</i> |
| i. A typical example. | i. A typical example. | i. A typical example. |
| j. A typical example. | j. A typical example. | j. A typical example. |
| k. A typical example. | k. A typical example. | k. A typical example. |
| l. A typical example. | l. A typical example. | l. A typical example. |
| m. A typical example. | m. A typical example. | m. A typical example. |
| n. A typical example. | n. A typical example. | n. A typical example. |

If the labels are numeric, label alignment can have an even bigger effect. Again, individual tastes and publishers' demands may differ. My preference is right alignment in this case.

- | | | |
|---------------------------------|-----------------------------------|----------------------------------|
| (35) <i>left aligned labels</i> | (36) <i>center aligned labels</i> | (37) <i>right aligned labels</i> |
| 7. A typical example. | 7. A typical example. | 7. A typical example. |
| 8. A typical example. | 8. A typical example. | 8. A typical example. |
| 9. A typical example. | 9. A typical example. | 9. A typical example. |
| 10. A typical example. | 10. A typical example. | 10. A typical example. |
| 11. A typical example. | 11. A typical example. | 11. A typical example. |
| 12. A typical example. | 12. A typical example. | 12. A typical example. |

The initial setting is left alignment for letters (either uppercase or lowercase) and right alignment for numbers. Unless numbers or letters of significantly different widths appear as labels, most users will not notice the difference and can safely ignore the issue.

It is a side issue, but the reader may have wondered how (32–34) and (35–37) were typeset. The idea is simple. You say:

```
\line{\divide\hsize by 3
  \vbox{\pex ... \xe}\hss
  \vbox{\pex ... \xe}\hss
  \vbox{\pex ... \xe}}
```

`\hss` is used to give a little stretch or shrink so that dimensional rounding does not lead to an under or overfull `\line{...}`. It is important in examples like this that `\pexcnt` is incremented globally, so that the latter `vboxes` see the `\excnt` which results from an operation inside a previous `vbox`. We will see later that in some situations this behaviour is not desirable and how it can be altered.

Variations are useful. One can easily imagine a situation in which something like the following is appropriate for two side by side examples.

```
\line{%
  \vbox{\hsize=.55\hsize \pex ... \xe}\hss
  \vbox{\hsize=.45\hsize \pex ... \xe}}
```

6.4. Relative versus fixed dimensions

Tex has two kinds of dimensional units. Dimensions specified in term of “pts”, “inches”, “cm”, etc. are fixed. Dimensions specified in terms of the units “em” or “ex” are relative to the particular text font that is current. Historically, an em is the width of a capital M and an ex is the height of a lowercase x. This is still more or less true, but each font is free to specify the equivalents in any way that it sees fit. The difference has important implications for parameter setting. A value specified in terms of em or ex units can be used without change in both the main text and footnotes, for example. If we set `textoffset=1em` at the beginning of a document, the proportions will stay the same whether we used 10pt or 12pt type, or 8 or 9 pt type in footnotes. If the document is set in 12pt type, and 1em is specified to be 12pt in that main text font, it makes no difference for typesetting in the main text font whether we set `textoffset=1em` or `textoffset=12pt`. But it does make a difference in sections of the document where a font with a different em dimension is used. If we switch to a font with a 10pt em unit, then the first specification will give a physical offset of 10pt, but the second will give a physical offset of 12pt.

When a dimension or skip parameter is reset by incrementing the old value, the new value is specified as a fixed dimension, which will not scale with font changes. The same is true of setting lengths indirectly by setting `labelwidth` or `exnwidth`. Adjusting a parameter by incrementing the old value, or using `samplelabel` or `exnwidth` to set the width of the label slot or the effective width of the example number, should only be used to make local adjustments, not at a level which has font size changes in its scope. For this reason, *expex.tex* specifies the label width in the label type `alpha` by `labelwidth=.72em`, not by `samplelabel=a`.⁴ Setting `labelwidth` via the second method would only be satisfactory if the font in force at the points that `\pex` is used is the same as the font in force when *expex.tex* is input and the `alpha` style defined.

6.5. User designed labeling

Macro: `\definelabeltype`

Counter: `\pexcnt`

Parameters:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|------------------------|------------------------------------|----------------------|
| <code>labelgen</code> | char, number, romannumber, or list | char |
| <code>pexcnt</code> | integer | 97 |
| <code>labellist</code> | comma separated list | {} |

If `labeltype` is set to `alpha`, the counter `\pexcnt` is set to 97, the character code of lowercase a in standard roman font sets, and `labelgen` is set to `char`. The successive labels are generated by taking the character corresponding to `\pexcnt` and stepping the counter by 1. This relies on the fact that the alphabetical sequence of characters corresponds to the numerical order of the character codes of the characters. Setting `labeltype` to `caps` is almost the same, except that `\pexcnt` is initialized to 65, the character code of uppercase A. If `labeltype` is set to `numeric`,

4. In the font that I happen to be using to write this section, the alphabetical labels (including the period) a–d have widths varying from .694em to .75em; the capital labels A–D have widths varying from .917em to .972em; and the integer labels for 1 to 9 all have width .75em. The label width settings for the label types `alpha` (.72em) and `caps` (.94em) are good compromises.

`labelgen` is set to `number` and `\pexcnt` is initialized to 1. The labels are generated by taking the number corresponding to the value of `\pexcnt`. If `labeltype` is set to `roman`, `\pexcnt` is initialized to 1 and `labelgen` is set to `romannumber`. All of these predefined label types also set the label format, alignment, and width appropriately. See *expex.tex* for examples of the use of `\definelabeltype`.

Something like the following might be useful. In the format I am using, `\mit` selects a math italics font which has the lowercase greek letters starting with the position 11.

```
\pex[labelgen=char,pexcnt=11,
everylabel=\mit]
\ a
\ a
\ a
\ a
\ a
\ x e
```

| | |
|------|------------|
| (38) | α . |
| | β . |
| | γ . |
| | δ . |

This scheme has a quirk if labels above ξ are needed because the correspondence between numerical order and alphabetical order breaks down at this point. o will be missing, with π following ξ .

This labeling scheme can be defined by:

```
\definelabeltype{greekmath}{labelgen=char,pexcnt=11,everylabel=\mit,
labelformat=A.}
```

Then `\pex[labeltype=greekmath]` is sufficient to invoke this labeling style.

If a sequence of character labels is needed which does not appear in sequence in a font, it is necessary to generate the labels from a list. Documents in Greek, for example, face the following problem. Roman letters were largely borrowed from the Greek alphabet, but Greek alphabetical order was not. Common greek fonts place letters in the position of the borrowing, not in their natural order in the Greek alphabet. Documents written in Greek, therefore, will have to generate the labels in multipart examples from a list. The solution is to set `labelgen` to `list`, and `labellist` to the desired list of labels.

For example, suppose the label type `greek` is defined by

```
\definelabeltype{greek}{labelgen=list,
labellist={a,b,g,d,e,z,h,j,i,k,l,m,n,x,o,p,r,sv,t,u,f,q,y,w}}
```

and that `\gr` selects one of the `grmn` series of fonts in the `cb` family of Greek fonts.

```
\gr
\pex[labeltype=greek]
\ a {\rm a,b,g,d,e,z}
\ a a,b,g,d,e,z
\ a {\rm a,b,c,d,e,f}
\ a a,b,c,d,e,f
\ a {\rm A,B,G,D,E,Z}
\ a A,B,G,D,E,Z
\ x e
```

| | | |
|------|--------------|--|
| (39) | α . | a,b,g,d,e,z |
| | β . | $\alpha,\beta,\gamma,\delta,\epsilon,\zeta$ |
| | γ . | a,b,c,d,e,f |
| | δ . | $\alpha,\beta,\zeta,\delta,\epsilon,\varphi$ |
| | ϵ . | A,B,G,D,E,Z |
| | ζ . | A,B, Γ , Δ ,E,Z |

“sv” appears in the list rather than “v” so that a nonfinal sigma is produced rather than a final sigma. They differ. “v” produces what amounts to a vertical strut of zero width, making the sigma nonfinal, but contributing no visible material. (Thanks to Christos Vlachos for this idea.)

Of course, if the entire document is written in Greek, then

```
\lingset{labeltype=greek}
```

should have global scope, so that parameter setting is not necessary in each `\pex` construction.

6.6. The parameter `sampleexno`

Parameter:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|-------------------------|--------------|----------------------|
| <code>sampleexno</code> | token list | {} |

In some publications, if two examples are close together in the running text and the widths of the typeset example numbers are different, the offsets are modified so that the texts in the two examples are aligned. The following is considered, under this stringent aesthetic, to be less than ideal.

- (9) a. I consider firemen available. (generic only)
b. I consider firemen intelligent. (generic only)

Exceptional case marking (ECM) verbs seem more or less to allow both existential and generic interpretations of complement subjects:

- (10) a. I believe firemen to be available. (both generic and existential)
b. I believe violists to be intelligent. (generic only)

`ExPex` provides the parameter `sampleexno` to handle this formatting problem. If the parameter is set to the empty token list, it has no effect of the formatting. If it is set to a nonempty token list, that token list is put in an hbox and its width is taken to be the effective typeset width of the example number.

- (9) a. I consider firemen available. (generic only)
b. I consider firemen intelligent. (generic only)

Exceptional case marking (ECM) verbs seem more or less to allow both existential and generic interpretations of complement subjects:

- (10) a. I believe firemen to be available. (both generic and existential)
b. I believe violists to be intelligent. (generic only)

```

\pex[sampleexno=(10)]
\a I consider firemen available. (generic only)
\a I consider firemen intelligent. (generic only)
\xe
Exceptional case marking (ECM) verbs seem more or less to allow both
existential and generic interpretations of complement subjects:
\pex
\a I believe firemen to be available. (both generic and existential)
\a I believe violists to be intelligent. (generic only)
\xe

```

This kind of fine tuning should only be done at the immediately pre-publication point because it depends upon the final assignment of example numbers and an aesthetic judgement of when two multipart examples are “visually close” in the finished product.

It seems to be common for publishers to do fine tuning of this sort in footnotes. There are two reasons for this. First, lowercase roman numerals are commonly used and their widths vary noticeably. Second, only a few examples are involved, assuming that examples numbers in a footnote start at (i), so the final assignments of example numbers is relatively easy to determine. The role of `sampleexno` in footnotes is taken up in Section 6.8.

6.7. IJAL style format of multiline examples

Macro: `\actualexno`

Parameters:

| <i>key</i> | <i>value</i> | <i>initial value</i> | <i>default value</i> |
|---------------------------------|--------------|----------------------|----------------------|
| <code>avoidnumlabelclash</code> | boolean | <code>false</code> | <code>true</code> |
| <code>appendtopexarg</code> | token list | <code>{}</code> | |

The formatting demands of the International Journal of American Linguistics (IJAL) require some additional parametrization. Multipart examples look like this:

- (4a) first
- (4b) second
- (64) Preamble
- (64a) first
- (64b) second

(1025) Preamble

(1025a) first

(1025b) second

A first approximation is

```
\lingset{labelanchor=numleft,labeloffset=0pt,
  textanchor=normal,textoffset=1.8em,
  preambleanchor=text,preambleoffset=0pt,
  labelformat=(A),everylabel=\actualexno}
```

`\actualexno` expands to the numerical value in `\excnt`, provided no special example number is set by `exno`, otherwise to the special example number. But the results (below) have some major problems.

(5a) first

(5b) second

(65) Preamble

(65a) first

(65b) second

(102) Preamble

(102a) first

(102b) second

There are two things that need to be done. First, in order to avoid printing both the example number and the label of the first part in examples with no preamble, as is the case in (5), printing the example number must be suppressed if there is no preamble. Second, in order to avoid the shrinking gap between the label and the text as more digits appear in the example number and it gets wider, the label width must be made dependent on the example number.

`ExPex` provides the boolean parameter `avoidnumlabelclash` which, if set to `true`, suppresses printing the example number in `\pex` constructions *if there is no preamble*. It has the XKV default value `true`, so that it can be set by giving the key with no label. Although it has the default value `true`, it has the initial value `false`. So, with the parameters set as above, we get:

```
\pex[exno=5,avoidnumlabelclash]
\ a first
\ a second
\ xe
```

| |
|-------------|
| (5a) first |
| (5b) second |

In order to solve the problem of the shrinking gap between the label and text, we could try something like `samplelabel=(\actualexno a)`. But it is considerably less than elegant to have to write this in every `\pex` example. It will not produce the desired result to put `\lingset{samplelabel=(\actualexno a)}` at the beginning of the document. `labelwidth` will be set to whatever the current width of `(\actualexno a)` is, which is not likely to be what you want for the entire document even if `\actualexno` happens to be defined at the point that

the `\lingset` is executed. To circumvent the global/local problem, *ExPex* provides the parameter `appendtopexarg`. Its value (unexpanded) is appended to whatever other arguments are given to `\pex` and evaluated locally.

```
\lingset{appendtopexarg={samplelabel=(\actualexno a)}}
```

accomplishes what we want.

We can summarize the discussion by defining the IJAL style.

```
\definelingsstyle{IJAL}{labelwidth=2em,labelanchor=numleft,
  labeloffset=0pt,labelformat=(A),everylabel=\actualexno,
  textanchor=normal,textoffset=1em,preambleanchor=text,
  preambleoffset=0pt,avoidnumlabelclash,
  appendtopexarg={samplelabel=(\actualexno a)}}
```

Then

```
\lingset{lingsstyle=IJAL}
\pex[exno=5]
\ a first
\ a second
\ xe
```

```
\pex~[exno=65]
Preamble
\ a first
\ a second
\ xe
```

```
\pex~[exno=1026]
Preamble
\ a first
\ a second
\ xe
```

| | |
|---------|----------|
| (5a) | first |
| (5b) | second |
| (65) | Preamble |
| (65a) | first |
| (65b) | second |
| (1026) | Preamble |
| (1026a) | first |
| (1026b) | second |

6.8. Footnotes and endnotes

Macro: `\keepexcntlocal`

Footnotes and endnotes pose a somewhat thorny problem since many different *Tex* and *LaTeX* macros are used to typeset footnotes and endnotes. Further, there are different ways of assigning example numbers and labels in multipart examples in footnotes. The footnote referenced at the end of this sentence is representative.⁵ It is an abbreviated version of footnote 17 in Chapter 2 of Diesing's *Indefinites*, MIT Press. This footnote style is fairly common.

The *ExPex* distribution contains two files, *eptexfn.tex* and *epltexfn.sty* which may be helpful in producing footnotes in this style. They do two things. First, they each define a macro

5. The existential reading does not seem to be available for subjects of small clause complements of *consider*:

(`\everyfootnote`) which, if evaluated at the start of processing a footnote, ensures that examples are correctly formatted. Second, they each perform surgery on a standard `\footnote` macro so that (among other things) `\everyfootnote` is inserted in the appropriate place.

I intend these macro files both to be used directly or to serve as models for the creation of variations which satisfy user needs. To the latter end, I will go through *epltxfn.sty* and explain how it works. The file listing is:

```
1 \makeatletter
2 \def\everyfootnote{%
3   \keepexcntlocal
4   \excnt=1
5   \lingset{exskip=1ex,exnotype=roman,sampleexno=,
6     labeltype=alpha,labelanchor=numright,labeloffset=.6em,
7     textoffset=.6em}
8 }
9 \renewcommand{\@makefntext}[1]{%
10  \everyfootnote
11  \parindent=1em
12  \noindent
13  \@thefnmark.\enspace #1%
14 }
15 \resetatcatcode
```

Lines 2–8 define the macro `\everyfootnote` which will be inserted into the footnote macro. It first ensures that changes in `excnt` that are made in the footnote are kept local to the footnote. `\keepexcntlocal` is an *ExPex* macro whose execution causes changes in `\excnt` to be kept local to the group in which `\keepexcntlocal` is executed. Without `\keepexcntlocal`, changes in `excnt` inside a footnote would be visible outside the footnote group in which the changes occur. `excnt` is then initialized. Finally, parameters are set which control the formatting of examples in the footnote. The LaTeX `\footnote` command uses `\@makefntext` to typeset the footnote. It is defined in the `cls` file which is used. The redefinition in lines 9–14 is a modification of the `\@makefntext` macro defined in *article.cls*. It first executes `\everyfootnote`, then prints the footnote number flush left at full footnote size, not as a superscript.

Assuming that `\usepackage{epltxfn}` has been executed, the following code produces fn. 5.

-
- (i) a. I consider firemen available. (generic only)
 - b. I consider firemen intelligent. (generic only)

Exceptional case marking (ECM) verbs seem more or less to allow both existential and generic interpretations of complement subjects:

- (ii) a. I believe firemen to be available. (both generic and existential)
- b. I believe violists to be intelligent. (generic only)

```

\footnote{%
\lingset{sampleexno=(ii)}
The existential reading does not seem to be available for subjects of
small clause complements of {\it consider\}/:
\pex
\!a I consider firemen available. (generic only)
\!a I consider firemen intelligent. (generic only)
\!xe
Exceptional case marking (ECM) verbs seem more or less to allow both
existential and generic interpretations of complement subjects:
\pex
\!a I believe firemen to be available. (both generic and existential)
\!a I believe violists to be intelligent. (generic only)
\!xe
}

```

The code in *eptexfn.tex* is somewhat more complicated because *Plain Tex* does not number footnotes and uses the same font for running text and footnotes. `\everyfootnote` is identical. If it is compared with the footnote macros in the *TexBook* (p. 363), on which it is modeled, it is easy to see the significance of the various modifications.

I anticipate that although the footnote macros in *epltxfn.sty* and *eptexfn.tex* will be useful to some readers without modification, other users will need to modify them for one reason or the other. Endnotes, in particular, will require work. I hope that these two files will serve as useful models. Of course, trivial modifications of the various dimensions can be done easily. More extensive modification require appropriate levels of expertise. Users should feel free to write to me directly (j.frampton@neu.edu) or post questions to the *Ling-Tex* discussion group, or to send me modifications that might prove useful to others. A file *epltx-endnotes.sty*, for example, would be useful.

7. User defined styles

Macro: `\definelingstyle`

Parameters:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|------------------------|----------------------------------|----------------------|
| <code>everyex†</code> | token list | {} |
| <code>Everyex†</code> | token list | {} |
| <code>lingstyle</code> | list of key settings (see below) | {} |

Aside from all the *ExPex* parameters introduced to this point, there are numerous *Tex* parameters that affect the appearance of examples: line spacing, hsize, font selection, etc. These are all quantities that *Tex* itself provides mechanisms for setting to suit the user. You may want to make adjustments of these parameters every time a particular kind of example is typeset, so that special settings hold inside these examples independently of the contextual settings. For example, suppose you have many examples that, for some reason, you want to format like (40), with a narrow width,

italic font, an oversized gap between the example number and the text, and a somewhat greater than normal separation between baselines.

| |
|--|
| (40) <i>Und hier können wir sehen was für Unfug wird gemacht wenn er einen ganz langen Satz binnen kriegt.</i> |
|--|

One way to do this is:

```
\ex[textoffset=3em]
\hsize=3in \rightskip=0pt plus 2em \it \advance\baselineskip by 2pt
Und hier können wir sehen was für Unfug wird gemacht
wenn er einen ganz langen Satz binnen kriegt.
\xe
```

Some rightskip is included because with such a narrow width it is otherwise difficult to avoid overfull lines.

If you have many examples that you want to typeset in this way that are scattered throughout a document, it is awkward to have to remember all of these settings and enter them each time an example of this kind needs to be formatted. Furthermore, if you change your mind about some detail of this special formatting, you need to go through the document and change each instance of this formatting. *Expex* allows you to package all the formatting changes into one named unit, called a “style”. If this style is called “narrow italic”, then you can write the following to achieve the format of (40).

```
\ex[lingstyle=narrow italic]
```

The various format specifications are packaged into a style by saying:

```
\definelingstyle{narrow italic}{textoffset=3em,
everyex={\hsize=3in \rightskip=0pt plus 2em
\it \advance\baselineskip by 2pt}}
```

Thereafter, setting the parameter `lingstyle` to `narrow italic` has the effect of setting all the parameters as specified in the style definition (the two parameters `textoffset` and `everyex` in this example). The parameter `everyex` works in the following way. If `everyex` is set to *value*, a macro `\lingeveryex` is defined whose expansion is *value*. When an `\ex` or `\pex` construction is initiated, *after* any parameter settings take effect, `\lingeveryex` is executed.

Expex provides a second parameter, `Everyex`, which is similar to `expex`. If `Expex` is set to *value*, a macro `ling@Everypex` is defined. When an `\ex` or `\pex` construction is initiated, *before* after any local parameter changes take effect, `\lingEveryex` is executed. The reasons for providing both `everyex` and `Everyex` are subtle. Different stages of the writing/rewriting/editing process may call for different treatments of example formatting, particularly if the final aim is a camera ready product. Suppose you normally make double spaced drafts with an `hsize` of 6.5in and that your final aim is to produce camera ready copy with an `hsize` of 4.375in. Suppose also that you are at editing stage where you want to see exactly how the examples will be formatted in

the finished product, but still want full width double spaced text. One way to accomplish this is to say

```
\lingset{everyex={\hspace=4.375in \normalbaselines}}
```

at the beginning of your document. `\normalbaselines` is a standard *TeX* macro which establishes the normal line spacing for the current font. This makes the `hsize` and line spacing inside examples independent of the `hsize` and line spacing you select for the text (outside examples).

This will accomplish what you want. But it makes the further use of `everyex` in your document awkward. If you want a particular example to be set in italics, for example, you might think to use:

```
\ex[everyex=\it]
```

This will certainly produce an italicized example because `\it` will be evaluated early on in typesetting the example. But it has the unfortunate consequence of overwriting the initial setting of `everyex`, removing the special line spacing within examples.

`Everyex` is provided to accommodate situations like this. The intention is that special settings that should hold inside examples throughout the document are assigned to `Everyex`, with `expex` reserved for local variations. Since `\lingEveryex` is evaluated before local parameter changes take effect, parameter settings specified by `Expex` will be overridden by any local parameter settings.

8. Judgment marks

Macros: `\judge`, `\ljudge`

Parameter:

| <i>key</i> | <i>value</i> | <i>initial value</i> | <i>default value</i> |
|------------|------------------------|----------------------|----------------------|
| * | sample judgment string | * | * |

In examples without parts, not much needs to be said.

```
\ex *Jack and Jill wented up the hill.\xe
```

(41) *Jack and Jill wented up the hill.

In my view (42), with a little whitespace inserted between the asterisk and the example sentence, looks somewhat better than (41), but the difference is slight.

```
\ex \judge* Jack and Jill is going up the hill.\xe
```

(42) *Jack and Jill wented up the hill.

`Expex` provides the macro `\judge` to accomplish this. `\judge` takes one argument. A multi-character judgment diacritic therefore needs to be surrounded in braces. `\judge` also ignores

following spaces. So `\judge{??}Mary...` and `\judge{??} Mary...` produce the same thing, as do `\judge*Mary...` and `\judge* Mary...`

Multipart examples are more complex, if alignment is to be maintained. If you find (43) satisfactory, what follows will not be of much interest. But if you would like the text (not including the judgment marks) to be aligned, read on.

- | |
|--|
| (43) a. There is a pair of pants on the floor. b. ?*There are a pair of pants on the floor. c. *There is the pair of pants on the floor. |
|--|

ExPex provides the macro `\ljudge` which pushes the judgment diacritics into the gap between the labels and the examples, instead of pushing the examples to the right to make room for the judgment diacritics. So

```
\pex
\a There is a pair of pants on the floor.
\a \ljudge{?*}There are a pair of pants on the floor.
\a \ljudge*There is the pair of pants on the floor.
\xe
```

produces

- | |
|---|
| (44) a. There is a pair of pants on the floor. b?*There are a pair of pants on the floor. c.*There is the pair of pants on the floor. |
|---|

Unfortunately, depending on the setting of `textoffset`, there is unlikely to be sufficient room for judgment diacritics in between the labels and the examples. `textoffset` needs to be increased to make room.

ExPex provides the pseudo parameter to facilitate adjusting the text offset. The key name is `*`, which is succinct, but perhaps overly so. In the setting below, the `*` on the left side is the parameter and `*` on the right side is a sample judgment mark. The setting below increments the text offset by the width of `'?*'`.

```
\pex[*=?*]
\a There is a pair of pants on the floor.
\a \ljudge{?*}There are a pair of pants on the floor.
\a \ljudge*There is the pair of pants on the floor.
\xe
```

- | |
|--|
| (45) a. There is a pair of pants on the floor. b. ?*There are a pair of pants on the floor. c. *There is the pair of pants on the floor. |
|--|

`textoffset` is increased by the width of the judgment diacritic which is furnished as the value of the parameter `*`.

If you say `\lingset{*}`, with no value assigned to `*`, it is given a default value, which happens to be `*`. So `\lingset{*}` is equivalent to `\lingset{*=*}`. So, for example:

```
\pex[*]
\a There is a pair of pants on the floor.
\a \ljudge* There are a pair of pants on the floor.
\a \ljudge* There is the pair of pants on the floor.
\xe
```

- (46) a. There is a pair of pants on the floor.
 b. *There are a pair of pants on the floor.
 c. *There is the pair of pants on the floor.

If you think that the text offset in (45) is too large, `textoffset` can be further adjusted directly, so you could write

```
\pex[*=?*,textoffset=!-.3em]
\a There is a pair of pants on the floor.
\a \ljudge{?*} There are a pair of pants on the floor.
\a \ljudge* There is the pair of pants on the floor.
\xe
```

- (47) a. There is a pair of pants on the floor.
 b. ?*There are a pair of pants on the floor.
 c. *There is the pair of pants on the floor.

9. Glosses

Macros: `\beginl[]`, `\glpreamble[]`, `\gla[]`, `\glb[]`, `\glc[]`, `\glft[]`, `\endgl`

Before introducing the many parameters which control the visual characteristics of glosses (fonts, line spacing, etc.), we first consider some simple glosses. They are coded in a traditional coding syntax; another syntax for coding glosses will be discussed later.

(100) *k- wapm -a -s'i -m -wapunin -uk*
 CL V AGR NEG AGR TNS AGR
 2 see 3ACC 2PL preterit 3PL
 'you (pl) didn't see them'

```

\ex
\beginl
\gla k- wapm -a -s'i -m -wapunin -uk //
\glb CL V AGR NEG AGR TNS AGR //
\glb 2 see {\sc 3acc} {} {\sc 2pl} preterit {\sc 3pl} //
\glft 'you (pl) didn't see them'//
\endgl
\xe

```

(101) Mary ist sicher, dass es den Hans nicht stören würde seiner Freundin ihr Herz auszuschütten.

Mary_i ist sicher, dass es den Hans nicht stören würde seiner Freundin ihr_i Herz auszuschütten.
 Mary is sure that it the-ACC Hans not annoy would his-DAT girlfriend-DAT her-ACC heart-ACC out to throw

'Mary is sure that it would not annoy John to reveal her heart to his girlfriend.'

```

\ex \beginl
\glpreamble Mary ist sicher, dass es den Hans nicht st\oren w\"urde
seiner Freundin ihr Herz auszusch\"utten.//
\gla Mary$_i$ ist sicher, dass es den Hans nicht st\oren w\"urde
seiner Freundin ihr$_i$ Herz auszusch\"utten.//
\glb Mary is sure that it the-{\sc acc} Hans not annoy would
his-{\sc dat} girlfriend-{\sc dat} her-{\sc acc}
heart-{\sc acc} {out to throw}//
\glft 'Mary is sure that it would not annoy John to reveal her
heart to his girlfriend.'//
\endgl \xe

```

All line wrapping is automatic. If the hsize were 4 in, for example, the code above would produce (102).

(102) Mary ist sicher, dass es den Hans nicht stören würde seiner Freundin ihr Herz auszuschütten.

Mary_i ist sicher, dass es den Hans nicht stören
 Mary is sure that it the-ACC Hans not annoy
würde seiner Freundin ihr_i Herz
 would his-DAT girlfriend-DAT her-ACC heart-ACC
auszuschütten.
 out to throw

'Mary is sure that it would not annoy John to reveal her heart to his girlfriend.'

Glosses (`\begin{gls} ... \end{gls}`) have up to three parts, all of which are optional. They are illustrated below.

- (103) Mary ist sicher, dass es den Hans nicht stören würde seiner Freundin ihr Herz auszuschütten. } preamble
- Mary_i ist sicher, dass es den Hans nicht stören würde seiner Freundin ihr_i Herz auszuschütten.* } interlinear gloss
- would his-DAT girlfriend-DAT her-ACC heart-ACC
out to throw
- ‘Mary is sure that it would not annoy John to reveal her heart to his girlfriend.’ } free translation

The code for the interlinear gloss consists of a sequence of lines of the form

`\glslevelname ... //`

where *levelname* is a, b, or c.⁶ There must be one and only one `\glsa` line, which must come first in the interlinear gloss. `\glsb` and `\gls c` lines can come in any order and can be repeated arbitrarily.

The material delineated by `\glslevelname` and `//`, `\gls preamble` and `//`, or `glsft` and `//` is parsed as a sequence of space separated items. The parser only looks for spaces at the top-level. Consequently, in (102), for example, it is not sensitive to the space in items like `the-{\sc acc}` since the space is inside a group, therefore not at the top level. Spaces that directly precede terminating `//` are disregarded. If a line in the interlinear gloss has more items on it than the `gla` line, the excess items are discarded. If it has fewer items than the `gla` line, it acts as if it ended with empty `{}` items.

9.1. Parameters

It is easiest to understand what the parameters do by examining how a gloss display is constructed. Consider (104), for example, in which box outlines have been added to facilitate discussion.

- (104) Mary ist sicher, dass es den Hans nicht stören würde seiner Freundin ihr Herz auszuschütten.

| | | | | | | | | | | |
|-------------------------|------------------------|----------------|-----------------------|-----------|------------|-------------|--------------|---------------|--------------|---------------|
| <i>Mary_i</i> | <i>ist</i> | <i>sicher,</i> | <i>dass</i> | <i>es</i> | <i>den</i> | <i>Hans</i> | <i>nicht</i> | <i>stören</i> | <i>würde</i> | <i>seiner</i> |
| Mary | is | sure | that | it | the-ACC | Hans | not | annoy | would | his-DAT |
| <i>Freundin</i> | <i>ihr_i</i> | <i>Herz</i> | <i>auszuschütten.</i> | | | | | | | |
| girlfriend-DAT | her-ACC | heart-ACC | out to throw | | | | | | | |

‘Mary is sure that it would not annoy John to reveal her heart to his girlfriend.’

1. The preamble is typeset as ordinary running text.

6. This will be extended later to allow the user to define new level names.

2. All of the items in the various lines in the interlinear gloss are accumulated, then the boxes which make up the interlinear gloss are typeset. These vboxes, outlined in (104), will be called *glwords*. A strut is inserted on each line of each *glword*.⁷ As these boxes are generated, they are fed to *TeX*'s standard paragraph building machinery, with the interword space set to `Val{gl{space}}`.
3. Finally, the free translation is typeset as ordinary running text.

Before giving a systematic list of gloss parameters, an example will help clarify how the parameter system operates. Suppose the first line for the code to (100) is replaced by:

```
\ex[gl{space}=!1em, everygl{a}={}, everygl{b}=\footnotesize, abovegl{b}skip=-.2ex]
```

the display below is produced. Compared to (100), the space between *glwords* has been increased by 1 em, the *gla*-line is not longer italicized, the *glb*-line is set in a smaller font, and it is moved closer to the *gla*-line.

```
(105) k-   wapm   -a     -s'i   -m     -wapunin  -uk
      CL   V      AGR    NEG    AGR     TNS       AGR
      2    see   3ACC          2PL    preterit  3PL

      'you (pl) didn't see them'
```

The parameters `everygl{a}` and `everygl{b}` are what are called 'hooks' into the coding of the gloss. The value assigned to each of these parameters is stored as a list of tokens, and these tokens are inserted into the code at the appropriate place. The tokens `Val{everygl{a}}` are inserted at the start of each *gla*-line in every *glword* and the tokens `Val{everygl{b}}` are inserted at the start of each *glb*-line in every *glword*. `everygl{a}` is initially set to `\it`, so that `Val{everygl{a}}` is `{\it}` initially. There are a number of hooks of this type.

(106) Hooks

| | |
|--------------------------------|------------------------|
| | <i>initial setting</i> |
| <code>everygl</code> | <code>{}</code> |
| <code>everygl{preamble}</code> | <code>{}</code> |
| <code>everygl{lilg}</code> | <code>{}</code> |
| <code>everygl{a}</code> | <code>\it</code> |
| <code>everygl{b}</code> | <code>{}</code> |
| <code>everygl{c}</code> | <code>{}</code> |
| <code>everygl{ft}</code> | <code>{}</code> |

The tokens `Val{everygl{}}` have scope over the entire gloss, `Val{everygl{lilg}}` over the interlinear gloss, etc.

There are a number of parameters like `abovegl{b}skip`, which was mentioned above, which control the vertical spacing. They are all (!)skip parameters. Recall that an (!)parameter can be set either directly, or as an increment from the present value of the parameter.

7. It is possible to turn off strut insertion inside *glwords* by setting the boolean parameter `gl{struts}` to `false`. It is unlikely that you will use this feature, except perhaps for diagnosing a spacing problem. It was introduced as an aid in designing and building the *ExpEx* glossing machinery. It remains as a public option because it might be useful to someone at some point.

(107) Vertical spacing

| | <i>initial setting</i> |
|-----------------------------------|------------------------|
| <code>aboveglbskip†</code> | 0pt |
| <code>aboveglcskip†</code> | 0pt |
| <code>belowglpreambleskip†</code> | 1ex |
| <code>aboveglftskip†</code> | 1ex |
| <code>extraglskip†</code> | 0pt |

Except for `extraglskip`, the meanings should be clear. Vertical skip `Val(extraglskip)` is put between the lines of `glwords` in the interlinear gloss. The initial setting is `.5ex`, but increasing it sometimes makes the interlinear gloss significantly more readable.

Finally, there are parameters which control how the `glwords` are formed into a paragraph.

(108) Horizontal spacing and hanging indentation in the interlinear gloss

| | <i>initial setting</i> | |
|---------------------------|---------------------------------------|---|
| <code>glspace†</code> | <code>(!)skip</code> | <code>.5em plus .4em minus .15em</code> |
| <code>glrightskip</code> | <code>skip</code> | <code>0pt plus .1\hsize</code> |
| <code>glhangindent</code> | <code>dimension</code> | <code>1em</code> |
| <code>glhangstyle</code> | <code>none, normal, or cascade</code> | <code>normal</code> |

`Val(glspace)` is the horizontal skip between `glwords` and `Val(glrightskip)` is the right skip. The initial settings allow considerable stretch and some shrinkage in the space between `glwords` and up to 10% of the page width in whitespace at the right margin. This minimizes the chances of overfull lines and, since interlinear glosses generally have a somewhat ragged appearance, does not detract from their appearance. In unusual circumstances, narrow page width in particular, this may require some adjustment to avoid overfull lines. See Section 11.3.1 for further discussion.

The choice parameter `glhangstyle` specifies the kind of hanging indentation which is used in the interlinear gloss. The default is normal hanging indentation, as illustrated in (101), but there is a choice of eliminating hanging indentation. There is also the option of cascading hanging indentation. See Section 12.4 for an example of *cascading hanging indentation*.

9.1.1 Where should parameter changes be made?

Consider (105) again. The code can be written as follows, with each parameter setting taking the narrowest possible scope which achieves the intended effect.

```
\ex
\beginl[glSPACE=1.5em]
\gla[everygla=] k- wapm -a -s'i -m -wapunin -uk //
\glb[everyglb=\footnotesize,aboveglbskip=-.2ex]
  CL V AGR NEG AGR TNS AGR //
\glb 2 see {\sc 3acc} {} {\sc 2pl} preterit {\sc 3pl} //
\glft 'you (pl) didn't see them'//
\endgl
\xe
```

Narrow scope is conceptually satisfied, but from a coding standpoint it is more convenient to concentrate the settings in one place.

```

\ex[glSPACE=1.5em, everygla=, everyglb=\footnotesize, aboveglbSkiP=-.2ex]
\beginG
\gla k- wapm -a -s'i -m -wapunin -uk //
\glb CL V AGR NEG AGR TNS AGR //
\glc 2 see {\sc 3acc} {} {\sc 2pl} preterit {\sc 3pl} //
\glft 'you (pl) didn't see them'//
\endG
\Xe

```

This way of coding the gloss has the big advantage that it is much easier to see what the parameter settings are.

If you use multiple instances of the same gloss format, a style should be defined

```

\definelineGstyle{Potawatami}{glSPACE=1.5em, everygla=,
everyglb=\footnotesize, aboveglbSkiP=-.2ex}

```

Then, typesetting a gloss in that style is done simply.

```

\ex[lingstyle=Potawatami]
\beginG
:
\Xe

```

If you want a sequence of glosses to all be done in this style, you can say:

```

\beginGrouP
\lingset{lingstyle=Potawatami}
:
\endGrouP

```

9.2. Exceptional \gla items

Items on the \gla line are generally associated with items on the other lines of the interlinear gloss. There are however a few items, called here *exceptional items*, which are interpreted in an exceptional fashion. There are three kinds of exceptional items: the single character + or @, and any item which begins with the control sequence \nogloss.⁸

9.2.1 +

Sometimes it is desirable to override natural wrapping and break up the gloss so that the syntax is emphasized, as in the following.

8. Version 4 of *Expex* had two additional exceptional items, the single characters [and]. In version 5, these characters still operate as in Version 4, but their use is discouraged and their use as exceptional \gla characters will disappear from *Expex* at some point in the future. \nogloss is more general, as explained below.

(109) *Mary_i ist sicher,*
 Mary is sure
dass es den Hans nicht stören würde
 that it the-ACC Hans not annoy would
seiner Freundin ihr_i Herz auszuschiütten.
 his-DAT girlfriend-DAT her-ACC heart-ACC out to throw
 ‘Mary is sure that it would not annoy John to reveal her heart to his girlfriend.’

This is accomplished by inserting ‘+’ appropriately, as shown in the code below. When + is encountered, the line is broken and a new line started. Hanging indentation, either normal or cascading, is preserved.

```
\ex
\beginl
\gla Mary$_i$ ist sicher, + dass es den Hans nicht st\oren w\"urde
+ seiner Freundin ihr$_i$ Herz auszusch\"utten.//
\glb Mary is sure that it the-{\sc acc} Hans not annoy would
his-{\sc dat} girlfriend-{\sc dat} her-{\sc acc} heart-{\sc acc} {out to
throw}//
\glft ‘Mary is sure that it would not annoy John to reveal her
heart to his girlfriend.’//
\endgl
\xe
```

9.2.2 @

Sometimes it is desirable to omit the space between two entries. Below, the space between the prefix and the verb it inflects has been omitted.

(110) *wiye kepi e- ca*
 two whitemen 1P:3D-found

This is accomplished by inserting ‘@’ appropriately, as shown in the code below.

```
\ex
\beginl
\gla wiye kepi e- @ ca//
\glb two whitemen {\sc 1p:3d}- found//
\endgl
\xe
```

In the unlikely event that you need a gla entry which would normally be entered as @, enter it as `{\relax @}` so that it is not interpreted as a directive to omit a space.⁹ (111) below shows another use for the @ diacritic.

9.2.3 \nogloss

Suppose you want to produce a gloss display like the one below.

(111) Fa'nu'i yu' ni [[*O* tinaitai-mu *t*] na lepblu].
 show me Obl Op *WH*[obj].read-agr L book

Note that the brackets are not glossed.

This could be done using the @ gloss diacritic as follows.

```
\ex[everygla=,glhangstyle=normal]<@period>
\beginl
\gla Fa'nu'i yu' ni {[[] @ {\it O} t{\it in\}aitai-mu
{\it t\}} na {lepblu}.}//
\glb show me Obl {} Op {\it WH\}[obj].read-agr {} L book//
\endgl
\xe
```

The gloss alignment is preserved by the use of dummy empty elements in the second row.

If glosses are long and/or if there are many rows, it is tedious and error prone to insert suitable empty elements.¹⁰ `\nogloss` is intended to simplify constructing glosses in which there are unglossed items on the `\gla` line. To produce (111), you can write:

```
\ex[everygla=,glhangstyle=normal]<@period>
\beginl
\gla Fa'nu'i yu' ni \nogloss{[[] @ {\it O} t{\it in\}aitai-mu
\nogloss{{\it t\}} na {lepblu}.}//
\glb show me Obl Op {\it WH\}[obj].read-agr L book//
\endgl
\xe
```

If you want extra space between the brackets and the words they bracket, the following is one way to achieve this.

(112) Um-äsudda' häm yan [i taotao [*O* ni si Juan ilek-ña nu guahu [malägu'
 agr-meet we with the person Op Comp the Juan say-agr Obl me agr.want
 gui [asuddä'-ña *t*]]].
 he *WH*[obl].meet-agr

9. This is a change. In version 4, `{{@}}` was suggested. This no longer works because various parsing steps strip away the grouping. `{{{{@}}}` does work, but `{\relax @}` is more straightforward.

10. Glosses of narratives can have hundreds of elements.

```

\ex[everygla=,glhangstyle=normal]
\beginl
\gla Um-\ "asudda' h\ "am yan \nogloss{${[\,,$} @ i taotao \nogloss{${[\,,$} @
{\it 0\}/} ni si Juan ilek-\~na nu guahu \nogloss{${[\,,$} @ mal\ "agu' gui
\nogloss{${[\,,$} @ asudd\ "a'-\~na \nogloss{{\it t\}/}$\,]]]}$.)//
\glb agr-meet we with the person Op Comp the Juan say-agr Obl me
agr.want he {\it WH\}/}[obl].meet-agr//
\endgl
\xe

```

The Chamorro examples (111) and (112) are from Chung (1998). The Potawatami example is from Halle and Marantz (1993). The German example in (101), and (120) in Section (11), are from Landau (2001). The Kiowa example (110) was contributed by Daniel Harbour.

10. Nlevel glosses; an alternate coding syntax

Macro: `\endpreamble`

Parameter:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|----------------------|-----------------------------|----------------------|
| <code>glstyle</code> | <code>wrap or nlevel</code> | <code>wrap</code> |

The last section assumed the default setting, ‘wrap’. The ‘nlevel’ gloss style produces identical displays, but they are coded differently. Compare the two ways of coding the display (113), which repeats (100).

```

(113) k- wapm -a -s'i -m -wapunin -uk
      CL V   AGR NEG AGR TNS   AGR
      2 see 3ACC      2PL preterit 3PL
      'you (pl) didn't see them'

```

```

\ex[glstyle=nlevel]
\beginl
k-[CL/2]
wapm[V/see]
-a[AGR/\sc 3acc]
-s'i[NEG]
-m[AGR/\sc 2pl]
-wapunin[TNS/preterit]
-uk[AGR/\sc 3pl]
\glft 'you (pl) didn't see them'
\endgl
\xe

```

```

\ex[glstyle=wrap]
\beginl
\gla k- wapm -a -s'i -m
      -wapunin -uk //
\glb CL V AGR NEG AGR TNS AGR //
\glc 2 see {\sc 3acc} {}
      {\sc 2pl} preterit {\sc 3pl} //
\glft 'you (pl) didn't see them'//
\endgl
\xe

```

The advantage to this style of coding complex glosses is that the gloss of a word and the word itself are adjacent in the code, just as they are in the display which is produced; vertically adjacent in the display, horizontally adjacent in the code. This makes the code much more readable. In effect, an aspect of WYSIWG is built into the coding. This is particularly useful if the gloss has many words, the gloss of a narrative for example.

Putting aside consideration of the preamble and the free translation for the moment, the list between `\beginl` and `\endl` is processed as a space separated list. Only top level spaces are separators. Spaces inside [...] are effectively hidden from this parsing. So, for example, the space in `-a[AGR/\sc 3acc]` does not mislead the parser. The material inside [...] is processed as a / separated list. Of course, / in this material must be hidden from the parser so / cannot appear at the top level. The same is true of], for obvious reasons.

The glossed words are written on separate lines above, but this is only for clarity. The code below is equivalent. It saves on virtual paper, but is not as easily deciphered.

```
\ex[glstyle=nlevel]\beginl k-[CL/2] wapm[V/see] -a[AGR/\sc 3acc]
-s'i[NEG] -m[AGR/\sc 2pl] -wapunin[TNS/preterit] -uk[AGR/\sc 3pl] \glft
'you (pl) didn't see them'\endl\xe
```

The preamble and free translation in the wrap style are terminated by //. In the nlevel style, `\endpreamble` ends the preamble and `\endl` terminates the free translation. There is no special termination of the free translation. This is illustrated by the coding of (101) in the nlevel style.

```
\ex[glstyle=nlevel]
\beginl
\glpreamble Mary ist sicher, dass es den Hans nicht st\"oren w\"urde
seiner Freundin ihr Herz auszusch\"utten.\endpreamble
Mary$_i$[Mary]
ist[is]
sicher,[sure]
dass[that]
es[it]
den[the-\sc acc]
Hans[Hans]
nicht[not]
st\"oren[annoy]
w\"urde[would]
seiner[his-\sc dat]
Freundin[girlfriend-\sc dat]
ihr$_i$[her-\sc acc]
Herz[heart-\sc acc]
auszusch\"utten.[out to throw]
\glft 'Mary is sure that it would not annoy John to reveal her heart to
his girlfriend.'
\endl
\xe
```

10.1. Parameters which modify particular lines

The various lines in a gloss which is coded in the wrap style are identified by name, so parameters like `everygla` and `aboveglbskip` can be used to modify the named line. `everygla` modifies the `\gla` line and `aboveglbskip` modifies the `\glb` line. This method for modifying lines is not available for coding in the `nlevel` style because the lines are not numbered. Instead, they must be referred to positionally.

Parameters:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|-------------------------------|---------------------|----------------------|
| <code>glneveryline</code> | list of token lists | <code>{\it}</code> |
| <code>glnabovelineskip</code> | list of dimensions | <code>{}</code> |

| |
|---|
| (114) <code>k- wapm -a -s'i -m -wapunin -uk</code> CL V AGR NEG AGR TNS AGR 2 see 3ACC 2PL PRETERIT 3PL |
|---|

```
\ex[glstyle=nlevel,glneveryline={\it,\sc,\sc},
  glnabovelineskip={,-2pt}]
\beginl
k-[cl/2]
wapm[v/\rm see]
-a[agr/3acc]
-s'i[neg]
-m[agr/\sc 2pl]
-wapunin[tns/preterit]
-uk[agr/3pl]
\endgl
\xe
```

Note that a list of dimensions `{,-2pt}` with dimensions missing is acceptable; the missing dimensions are assumed to be `0pt`. Missing entries in the value of `glneveryline`, either because the list is shorter than the number of lines in the interlinear gloss or because of null entries on the list, are similarly assumed to be null and cause no problem. The material inside the brackets, or material delimited by `/` can be missing as well, but the brackets are mandatory. Spaces after `[` or `/` are ignored, so for example, `wapm[v/\rm see]` and `wapm[v/ \rm see]` produce the same output.

There is no parameter `aboveglaskip` for use in the wrap coding style since vertical skip in the interlinear gloss makes sense only between lines. Similarly, the first item in the list specified by `glnabovelineskip`, which corresponds to `\aboveglaskip`, is ignored.

10.2. \nogloss and the diacritics @ and +

A word gloss $\langle word \rangle[\langle gloss \rangle / \dots]$ can optionally be immediately followed by one of @ and +. A @ cancels the usual space between the typeset word glosses. A + introduces a line break. This should be compared with the different syntax in the wrap style; see Sections 9.2.2 and 9.2.1. In the nlevel style they are diacritics; in the wrap style they are exceptional words. For example, the following code produces the same output as the code which is given for (110).

```
\ex[glstyle=nlevel]
\beginogl wiye[two] kepi[whitemen] e-[\sc 1p:3d-]@ ca[found] \endogl
\xe
```

\nogloss in nlevel style glosses works very much the same way that it works in wrap style glosses.

| |
|--|
| (115) Fa'nu'i yu' ni [[<i>O tinaitai</i> -mu <i>t</i>] na lepblu]. show me Obl Op <i>WH</i> [obj].read-agr L book |
|--|

```
\ex
\beginogl[glstyle=nlevel,glneveryline={}]
Fa'nu'i[show]
yu'[me]
ni[Obl]
\nogloss{[[\thinspace]@ {\it O}[Op]
t{\it in\/}aitai-mu[{\it WH\/}{[obj]}.read-agr]
\nogloss{{\it t}\thinspace ]}
na[L]
lepblu[book]@ \nogloss{[.]}.}
\endogl
\xe11
```

10.3. Line spacing inside glwords

One spacing problem that is handled automatically in the wrap style must be handled by the user in the nlevel style. It is not common, but is worth mentioning because it might cause a perplexing problem if it does arise. Characters on lower levels which are extra tall can cause misalignment of the baselines. Suppose, for example, that you want to use a gloss to give some information about the morphophonological derivation of a surface form, as is attempted in (116).

| |
|--|
| (116) <i>m-</i> <i>wope</i> (mo-) (a \bar{w} ope) |
|--|

11. Italics are used to highlight an infix in the verb in the relative clause. The spacing could be improved. Horizontal skip should be used ; $\{\it \hspace{.8pt} in\hspace{.4pt}\}$, for example.

```
\ex[glstyle=nlevel,glneveryline={\it}]
\begin{gl} m-[(mo-)] wope[(a\AccentedBarredW ope)] \end{gl} \xe
```

Not only is more space between the lines needed, but there is misalignment of the baselines on the second level.

One might think to fix the problem in (116) by putting extra vertical skip between the lines of the gloss, as in the code below.

```
\ex[glstyle=nlevel,glneveryline={\it},glnabovelineskip={, .5ex}]
\begin{gl} m-[(mo-)] wope[(a\AccentedBarredW ope)] \end{gl} \xe
```

But the result is less than satisfactory:

(117) *m- wope*
(mo-) (a[˘]wope)

There is now vertical space over the second line of the glwords, but the baselines of the second lines of the glwords are not aligned. In wrap style coding, the material for all of the glwords is accumulated before any of the glwords are constructed, so the presence of an extra tall line in the second glword is known before the first glword is constructed. An adjustment is made to preserve baseline alignment. In nlevel style coding, the first glword is boxed and contributed to a paragraph before the material to construct the second glword is read, so there is no way for any adjustment to be made automatically.

There are two solutions. One solution for the problem above is to insert extra tall struts on the second line of every glword. Normal struts in the 12pt type used in this manual are 10pt high and 4pt deep. The following code will produce (118).

```
\ex[glstyle=nlevel,glneveryline={\it,\vrule height14pt width0pt}]
\begin{gl} m-[(mo-)] wope[(a\AccentedBarredW ope)] \end{gl}
\xe
```

(118) *m- wope*
(mo-) (a[˘]wope)

Another solution is to increase the baselineskip inside glwords.

```
\ex[glstyle=nlevel,glneveryline={\it},everyglword={\baselineskip=18pt}]
\begin{gl} m-[(mo-)] wope[(a\AccentedBarredW ope)] \end{gl} \xe
```

(119) *m- wope*
(mo-) (a[˘]wope)

11. More about glosses

11.1. The parameter `glwidth`

Parameter:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|----------------------|--------------|----------------------|
| <code>glwidth</code> | dimension | Opt |

If the parameter `glwidth` is set to a nonzero value, the gloss is built in a vbox whose width is the setting of `glwidth`. The following example illustrates the usefulness of the explicit width option.

- (120) a. *Mary_i ist sicher, dass es den*
 Mary is sure that it the-ACC
Hans nicht stören würde seiner
 Hans not annoy would his-DAT
Freundin ihr_i Herz
 girlfriend-DAT her-ACC heart-ACC
auszuschütten.
 out to throw
 ‘Mary is sure that it would not annoy John to reveal her heart to his girlfriend.’
- b. *Mary_i ist sicher, dass seiner*
 Mary is sure that his-DAT
Freunden ihr_i Herz
 girlfriend-DAT her-ACC heart-ACC
auszuchütten dem Hans nicht
 out to throw the-DAT Hans not
schaden würde.
 damage would
 ‘Mary is sure that to reveal her heart to his girlfriend would not damage John.’

```
\ex[glwidth=2.6in]
a.\quad
\begin{gl}
\gla Mary$_i$ ist sicher, dass es den Hans nicht st\oren w\urde
seiner Freundin ihr$_i$ Herz auszusch\utten.//
\glb Mary is sure that it the-{\sc acc} Hans not annoy would
his-{\sc dat} girlfriend-{\sc dat} her-{\sc acc} heart-{\sc acc} {out to
throw}//
\glft ‘Mary is sure that it would not annoy John to reveal her
heart to his girlfriend.’//
\end{gl}
\hfil
b.\quad
\begin{gl}
\gla Mary$_i$ ist sicher, dass seiner Freunden ihr$_i$ Herz
auszuch\utten dem Hans nicht schaden w\urde.//
\glb Mary is sure that his-{\sc dat} girlfriend-{\sc dat} her-{\sc acc}
heart-{\sc acc} {out to throw} the-{\sc dat} Hans not damage would//
\glft ‘Mary is sure that to reveal her heart to his girlfriend
would not damage John.’//
\end{gl}
\xe
```

11.2. Comments and citations

Macros: `\trailingcitation`, `\rightcomment`

Parameters:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|------------------------------------|--------------|----------------------|
| <code>mincitesep</code> | dimension | 1.5 em |
| <code>everytrailingcitation</code> | token list | empty |

The following illustrates two different ways to append a comment or citation to a gloss.

| | | |
|-------|---|------------------------|
| (121) | <i>k- wapm -a -s'i -m -wapunin -uk</i> | [Potawatami] |
| | Cl V Agr ₁ Neg Agr ₂ Tns Agr ₃ | category |
| | 2 see 3ACC 2PL preterit 3PL | |
| | 'you (pl) didn't see them' | (Hockett 1948, p. 143) |

```
\ex
\beginl
\gla \rightcomment{[Potawatami]}k- wapm -a -s'i -m -wapunin -uk //
\glb \rightcomment{category}Cl V Agr$_1$ Neg Agr$_2$ Tns Agr$_3$//
\glb 2 see {\sc 3acc} {} {\sc 2pl} preterit {\sc 3pl} //
\glft 'you (pl) didn't see them'\trailingcitation{(Hockett 1948,
  p. 143)}//
```

`\endgl`

`\xe`

or

```
\ex
\beginl[glstyle=nlevel,glneverylines={\it}]
{\rightcomment{[Potawatami]}k-}{\rightcomment{category}Cl/2}
wapm[V/see]
-a[Agr$_1$/\sc 3acc]
-s'i[Neg]
-m[Agr$_2$/\sc 2pl]
-wapunin[Tns/peterit]
-uk[Agr$_3$/\sc 3pl]
\glft 'you (pl) didn't see them'\trailingcitation{(Hockett 1948,
  p. 143)}
```

`\endgl`

`\xe`

`\trailingcitation` will put the citation on the same line as the last line of the free translation if there is enough room for it, otherwise at the end of the following line. The parameter `mincitesep` determines the minimum whitespace between the end of the free translation and the citation that

is tolerated; the default is 1.5em. `everytrailingcitation` can be used to specify the font for trailing citations.

`\trailingcitation` is useful outside the context of glosses.

- (122) a. [Which pilot who shot at it₁]₂ hit [which MIG₂ that had chased him₂]₁? (Barss, 2000)
 b. [Which pilot who shot at it₁]₂ hit [which MIG₂ that had chased him₂]₁?
 (Higgenbotham and May, 1981)

`\pex`

`\a\relax [Which pilot who shot at it$_1$]$_2$ hit [which MIG$_2$ that had chased him$_2$]$_1$?\trailingcitation{(Barss, 2000)}`

`\a\relax [Which pilot who shot at it$_1$]$_2$ hit [which MIG$_2$ that had chased him$_2$]$_1$?\trailingcitation{(Higgenbotham and May, 1981)}`

`\xe`

`\rightcomment{...}` must come first in the first item on the line. The macro is very primitive. It does not consider the width of the citation or the amount of whitespace at the right of the gloss. The citation will overlap the gloss if there is not room for it at the right. For example, if the gloss (122) is attempted with an `hsize` of 3.5in and `glspace=.5em`, the result is (123).

- (123) *k- wapm -a -s'i -m -wapm* [Dawa waka mi]
 Cl V Agr₁ Neg Agr₂ Tns category
 2 see 3ACC 2PL preterit 3PL
 'you (pl) didn't see them'
 (Hockett 1948, p. 143)

In spite of its limitations, `\rightcomment` is occasionally quite useful.

11.3. Line spacing in wrapped glosses

Users might prefer to increase `Val(extraglskip)`. Compare (124a), which uses the default setting of `extraglskip (.5ex)`, to (124b), which increases this by another `.5ex`.

- (124) a. *Um-äsudda' häm yan [i taotao [O ni si Juan ilek-ña nu guahu*
 agr-meet we with the person Op Comp the Juan say-agr Obl me
 [*malägu' gui [asuddä'-ña t]]*].
 agr.want he WH[obl].meet-agr
- b. *Um-äsudda' häm yan [i taotao [O ni si Juan ilek-ña nu guahu*
 agr-meet we with the person Op Comp the Juan say-agr Obl me
 [*malägu' gui [asuddä'-ña t]]*].
 agr.want he WH[obl].meet-agr

```

\pex[everygla=]
\begin{gla}
\gla Um-\asudda' h'am yan \nogloss{[\,$} @ i taotao \nogloss{[\,$}
@ {\it 0\} ni si Juan ilek-\na nu guahu \nogloss{[\,$} @ mal\agu'
gui \nogloss{[\,$} @ asudd"a'-\na \nogloss{{\it t\}\,$,)]}]$.}//
\glb agr-meet we with the person Op Comp the Juan say-agr Obl me
agr.want he {\it WH\}[obl].meet-agr//
\end{gla}
\begin{gla}[extraglskip=!.5ex] ... (same as the gloss above) \end{gla}
\end{pex}

```

11.3.1 How to avoid overfull boxes in glosses

Parameters:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|--------------------------|-------------------|----------------------|
| <code>glrightskip</code> | <code>skip</code> | 0pt plus .1 hsize |

Hopefully, you will never encounter the problem which the parameterization in this section is designed to solve, the “overfull line” error message. The default setting puts .5em of glue between the glwords on a line, plus .4em of stretchability and .15em of shrinkage. The interglword spacing on a line can therefore vary from .35em up to .9em to accommodate the needs of the Tex linebreaking algorithm.¹² Further, the default settings allow up to 10% of the hsize of whitespace to appear at the end of the line of glwords. The possible extra space at the right edge and stretchability/shrinkability of the space between glwords means that line breaking in glosses will rarely encounter problems with overfull lines.

In unusual cases, there may be a problem. Your publisher (or you) may be particularly fussy and demand a particular spacing and/or better right alignment in glosses. Or you might want to make an unusually narrow gloss, which increases the chances that there might not be enough flexibility in the spacing.

If you do run into the problem of overfull lines in a gloss, two parameters allow for a great deal of flexibility; `glspace` and `glrightskip`. `glspace` is an incremental parameter, so you could say, for example, `\lingset{glspace=!0pt plus .2em}`, increasing the stretchability of the interglword space by .2em. Or, you might want to increase the stretchability of the rightskip in a particular troublesome gloss to allow more whitespace at the right edge. An acceptable solution will depend on your particular typesetting aesthetics. Solving line breaking problems is often troublesome and requires experimentation.

12. This is a much larger range than is typical in running text, but it is appropriate in glosses, which typically have many large whitespace gaps on one line or the other.

12. Glosses; Special topics

12.1. User defined levels

Macro: `\defineglwlevels`

`\glb` and `\glc` are given definitions in *expex.tex* by the command `\defineglwlevels{b,c}`. The command also creates the parameters `everyglb`, `everyglc`, `aboveglbskip`, and `aboveglc-skip`. `everyglb` and `everyglc` are initialized to empty token lists and `aboveglbskip` and `aboveglc-skip` to 0pt. The user may want to use `\defineglwlevels` to create and name new gloss levels.

For example, suppose more suggestive level names are desired.

```
\defineglwlevels{cat,gloss}
\lingset{everyglcat=\footnotesize,aboveglcatskip=-.5ex}

\ex
\beginl
\gla k- wapm -a -s'i -m -wapunin -uk //
\glcat Cl V Agr Neg Agr Tns Agr //
\glgloss 2 see 3{\sc acc} {} {2\sc pl} preterit {3\sc pl} //
\glft 'you (pl) didn't see them'//
\endgl
\xe
```

produces

| | |
|-------|---|
| (125) | <i>k- wapm -a -s'i -m -wapunin -uk</i> |
| | Cl V Agr Neg Agr Tns Agr |
| | 2 see 3 _{ACC} 2 _{PL} preterit 3 _{PL} |
| | 'you (pl) didn't see them' |

Another instance in which the user might want to define a new gloss level or levels is if more than 3 lines of interlinear gloss are needed and the desired flexibility cannot be obtained by repeated use of `\glb` or `\glc`.

12.2. Positioning the free translation to the right of the interlinear gloss

Parameters:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|-------------|-------------------------|----------------------|
| glftpos | choice (below or right) | below |
| sssep | dimension | 3em |
| ssratio | decimal | .6 |
| ssrightskip | skip | 0pt plus 2em |

| | |
|---|---|
| <p>(126) <i>Homâo sa čô pô tha ñu nao ngă hmua. Ñu</i> EXIST one CLF person old 3s go do field 3s <i>djă gă, ñu djă čông ñu, lai h gui</i> hold machete 3s hold hoe 3s and carry.on.back <i>rêo ñu. Todang bboi rôk jolan ñu nao</i> back.basket 3s while at along trail 3s go <i>hma, ñu bbôh sa droi mră đố bboi gah,</i> field 3s see one CLF peacock stay at DRCT <i>a, hruh ñu.</i> – nest 3s</p> | <p>‘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.’</p> |
|---|---|

is achieved by

```
\ex[glftpos=right,glhangstyle=none]
\let\\=\textsc
\beginl
\gla
Hom\^{a}o sa \v{c}\^{o} p\^{o} tha \~{n}u nao ng\u{a} hmua. \~{N}u
dj\u{a} g\u{a}, \~{n}u dj\u{a} \v{c}\u{o}ng \~{n}u, lai h gui r\^{e}o
\~{n}u. Todang bboi r\^{o}k jolan \~{n}u nao hma, \~{n}u bb\^{o}h sa
droi mr\u{a} d\u{o} bboi gah, a, hruh \~{n}u.//
\glb
\\{exist} one \\{clf} person old \\{3s} go do field \\{3s} hold
machete \\{3s} hold hoe \\{3s} and carry.on.back back.basket \\{3s}
while at along trail \\{3s} go field \\{3s} see one \\{clf} peacock
stay at \\{drct} -- nest \\{3s}//
\glft
‘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.’//
\endl
\ex
```

(This example, as well as (127), was contributed by Joshua Jensen. It is from Jarai, an Austronesian language. The story teller was Hyech Ksor. The orthography here is somewhat simplified in order to keep the font requirements for the examples in this documentation elementary.)

`ss` stands for “side-by-side”. `sssep` gives the separation of the gloss and the free translation. `ssratio` gives the proportion of the available width that the gloss occupies. The point of hanging indentation is to visually separate the free translation and the gloss, so `glhangstyle=none` is completely satisfactory if the tree translation is on the right. But *ExPex* will happily use hanging indentation with the free translation on the right.

Line breaking in the free translation is delicate because it will generally set in a narrow column. The default setting of `ssrightskip` allows up to 2em departure from right alignment. This usually avoids overfull lines and awkward hyphenation. `ssrightskip` can be increased (all the way to 0pt plus 1 fil) if there is a problem, at the cost of a more ragged appearance. This can be done globally, or simply in troublesome examples.

12.3. Glosses with a side panel

Macros: `\beginlpanel[]`, `\endpanel`

Parameter:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|--------------------------|--------------|----------------------|
| <code>everypanel†</code> | token list | { } |

The mechanism for positioning the free translation to the right of the interlinear gloss can be adapted to create a side panel for glosses which can be used for other purposes, as illustrated below.

(127) *Homâo*¹ *sa* *čô* *pô* *tha* *ñu* *nao* *ngă*
 EXIST one CLF person old 3s go² do
hmua. *Ñu* *djă* *gă*, *ñu* *djă* *čöng*
 field 3s hold machete 3s hold hoe
ñu, *laih* *gui* *rêo* *ñu*.
 3s and³ carry.on.back back.basket 3s
Todang *bboi* *rôk* *jolan* *ñu* *nao* *hma*, *ñu*
 while at along trail 3s go field 3s
bbôh *sa* *droi* *mră* *dô*⁴ *bboi* *gah*, *a*,
 see one CLF peacock stay at DRCT –
hruh *ñu*.
 nest 3s

1. *homâo* also means ‘have’, reflecting the strong tendency across languages to use the same word for possession and the existential. *homâo* is clause-initial in existential clauses, but it comes after the subject in possession clauses.
2. All verbs are glossed with a bare form, as Jarai has no inflectional morphology. Although Jarai has lexical items that encode tense, they are relatively infrequent in text.
3. The word *laih* is literally ‘after; finish’, but that is clearly not the meaning here. Probably *laih* here is an abbreviation for *laih anăn*, ‘after that; and’, hence the gloss ‘and’.
4. *dô* ‘sit, stay’ is used like a copula in locative clauses, which is what I assume here (‘a peacock [which was] beside its nest’); however, this could just as well mean ‘a peacock sitting beside its nest’, retaining the posture semantics.

‘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.’

The syntax is:

```
\beginlpanel ... \endgl ... \endpanel
```

The first part is the gloss, with the usual syntax. The second part is put in a vbox and set alongside the gloss. The tokens `lingeverypanel` are inserted when the vbox begins. All of the parameters which are special to positioning the free translation to the right of the gloss apply here as well, with the obvious meanings.

The complete code for the example above is:

```
\ex[everypanel=\footnotesize]<panelex>
\let\l=\textsc
\beginlpanel[ssratio=.5,glhangstyle=none]
\gla Hom\^{a}o$^1$ sa \v{c}\^{o} p\^{o} tha \~{n}u nao ng\u{a}
hmua. \~{N}u dj\u{a} g\u{a}, \~{n}u dj\u{a} \v{c}\u{o}ng \~{n}u,
laih gui r\^{e}o \~{n}u. Todang bboi r\^{o}k jolan \~{n}u nao
hma, \~{n}u bb\^{o}h sa droi mr\u{a} d\u{o}$\,^4$ bboi gah, a, hruh
\~{n}u.//
\glb \{\exist} one \{\clf} person old \{\3s} go$^2$ do field
\{\3s} hold machete \{\3s} hold hoe \{\3s} and$^3$ carry.on.back
back.basket \{\3s} while at along trail \{\3s} go field \{\3s}
see one \{\clf} peacock stay at \{\drct} -- nest \{\3s}//
\endgl
```

1.\enspace `\it hom\^{a}o` also means ‘have’, reflecting the strong tendency across languages to use the same word for possession and the existential. `\it hom\^{a}o` is clause-initial in existential clauses, but it comes after the subject in possession clauses.

2.\enspace All verbs are glossed with a bare form, as Jarai has no inflectional morphology. Although Jarai has lexical items that encode tense, they are relatively infrequent in text.

3.\enspace The word `\it laih` is literally ‘after; finish’, but that is clearly not the meaning here. Probably `\it laih` here is an abbreviation for `\it laih an\u{u}n`, ‘after that; and’, hence the gloss ‘and’.

4.\enspace `\it d\u{o}` ‘sit, stay’ is used like a copula in locative clauses, which is what I assume here (‘a~peacock [which was] beside its nest’); however, this could just as well mean ‘a peacock sitting beside its nest’, retaining the posture semantics.

```
\endpanel
```

```
\bigskip
```

‘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.’

\xe

Note that the free translation here comes after \endpanel and is typeset the way any material inside an \ex construction is typeset. This allows it to have full width, spanning both the gloss and notes. It could have been part of the gloss, with a different result.

No support is given to side note numbering. It must be done “by hand”. If the construction turns out to be sufficiently useful and hand numbering is sufficiently tedious, a more automatic scheme might be possible. It would not be trivial, because the order in which the notes appear inside the gloss before it is typeset is not necessarily the same as the order in which they appear after it is typeset.

12.4. Cascading hanging indentation in glosses

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

is produced by

```
\ex[glhangstyle=cascade]
\let\\=\textsc
\beginl
\gla
Hom\^{a}o sa \v{c}\^{o} p\^{o} tha \~{n}u nao ng\u{a} hmua. \~{N}u
dj\u{a} g\u{a}, \~{n}u dj\u{a} \v{c}\u{o}ng \~{n}u, laih gui r\^{e}o
\~{n}u. Todang bboi r\^{o}k jolan \~{n}u nao hma, \~{n}u bb\^{o}h sa
droi mr\u{a} d\u{o} bboi gah, a, hruh \~{n}u.//
\glb
\\{exist} one \\{clf} person old \\{3s} go do field \\{3s} hold
machete \\{3s} hold hoe \\{3s} and carry.on.back back.basket \\{3s}
while at along trail \\{3s} go field \\{3s} see one \\{clf} peacock
stay at \\{drct} -- nest \\{3s}//
\glft
‘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
```


The introduction of the nlevel gloss style makes `\gluf` unnecessary. Its use should be avoided in the future because it will be removed from *ExPex* in some future version. The code below shows (129) would be coded in the nlevel style.

```
\ex[glstyle=nlevel,glneveryline={\it,,\footnotesize},
  glnabovelineskip={,,-.4ex},extraglskip=0pt]
\beginl
Mary$_i$[Mary]
ist[is]
sicher,[sure]
dass[that]
es[it]
den[the/ACC]
Hans[Hans]
nicht[not]
st\"oren[annoy]
w\"urde[would]
seiner[his/DAT]
Freundin[girlfriend/DAT]
ihr$_i$[her/ACC]
Herz[heart/ACC]
auszusch\"utten.[out to throw]
\glft ‘Mary is sure that to reveal her heart to his girlfriend
would not damage John.’
\endgl
\xe
```

The result is almost, but not quite, identical to (129). The underfixes are positioned somewhat differently, not center aligned under the word above them.

| | |
|-------|--|
| (130) | <i>Mary_i ist sicher, dass es den Hans nicht stören würde seiner Freundin ihr_i Herz</i> |
| | Mary is sure that it the Hans not annoy would his girlfriend her heart |
| | ACC DAT DAT ACC ACC |
| | <i>auszuschütten.</i> |
| | out to throw |
| | ‘Mary is sure that to reveal her heart to his girlfriend would not damage John.’ |

See (105) below for a gloss display in the nlevel style which is closer to (129).

12.6. Center alignment in glwords

Parameter:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|--------------------------|----------------|----------------------|
| <code>glwordalign</code> | left or center | left |

When glwords are assembled, in either the wrap or nlevel styles, an alignment inside a vbox is constructed. The alignments use an `\hfil` on the right, so that the text on the various lines of a glword is left aligned. The parameter `glwordalign` allows the choice of center alignment. This is occasionally useful. Compare, for example, (131) with (105).

| | | | | | | | |
|-------|----------------------------|------|------------------|------|-----------------|----------|-----------------|
| (131) | k- | wapm | -a | -s'i | -m | -wapunin | -uk |
| | CL | V | AGR | NEG | AGR | TNS | AGR |
| | 2 | see | 3 _{ACC} | | 2 _{PL} | preterit | 3 _{PL} |
| | ‘you (pl) didn’t see them’ | | | | | | |

```
\ex[glSPACE=1.5em, everygla=\hfil, glwordalign=center,
  everyglc=\hfil, aboveglbSKIP=-.2ex]
\beginGl
\gla k- wapm -a -s'i -m -wapunin -uk //
\glb CL V AGR NEG AGR TNS AGR //
\glc 2 see {\sc 3acc} {} {\sc 2pl} preterit {\sc 3pl} //
\glft ‘you (pl) didn’t see them’//
\endGl
\Xe
```

Center alignment in glwords is useful if you need to number the glwords; to facilitate discussion, for example.

| | | | | | | | | | | | | | | | |
|-------|---------------|--|-------------|-------------|---------------|-------------|-------------|---------------|-------------|--------------|--------------|------------|-------------|------------|------------|
| (132) | ¹³ | (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (9) | (10) | (11) | (12) | (13) | (14) |
| | | <i>Homâo</i> | <i>sa</i> | <i>čô</i> | <i>pô</i> | <i>tha</i> | <i>ñu</i> | <i>nao</i> | <i>ngă</i> | <i>hmua.</i> | <i>Ñu</i> | <i>djă</i> | <i>gă,</i> | <i>ñu</i> | <i>djă</i> |
| | | EXIST | one | CLF | person | old | 3s | go | do | field | 3s | hold | machete | 3s | hold |
| | | (15) | (16) | (17) | (18) | (19) | (20) | (21) | (22) | (23) | (24) | (25) | | | |
| | | <i>čông</i> | <i>ñu,</i> | <i>laih</i> | <i>gui</i> | <i>rêo</i> | <i>ñu.</i> | <i>Todang</i> | <i>bboi</i> | <i>rôk</i> | <i>jolan</i> | <i>ñu</i> | | | |
| | | hoe | 3s | and | carry.on.back | back.basket | 3s | while | at | along | trail | 3s | | | |
| | | (26) | (27) | (28) | (29) | (30) | (31) | (32) | (33) | (34) | (35) | (36) | (37) | (38) | |
| | | <i>nao</i> | <i>hma,</i> | <i>ñu</i> | <i>bbôh</i> | <i>sa</i> | <i>droi</i> | <i>mră</i> | <i>dô</i> | <i>bboi</i> | <i>gah,</i> | <i>a,</i> | <i>hruh</i> | <i>ñu.</i> | |
| | | go | field | 3s | see | one | CLF | peacock | stay | at | DRCT | – | 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.’ | | | | | | | | | | | | | |

```

\ex[glstyle=nlevel,glhangstyle=cascade,
    glneveryline={\insertno,\it,},glwordalign=center,
    glnabovelineskip={,-1pt},glspace=! .4em]
\beginl
(1)[Hom\^{a}o/\textsc{exist}]
(2)[sa/one]
(3)[\v{c}\^{o}/\textsc{clf}]
(4)[p\^{o}/person]
:
(35)[gah,/\textsc{drct}]
(36)[a,/--]
(37)[hruh/nest]
(38)[\~{n}u.\textsc{3s}]
\glft ...
\endgl
\xe

```

Numbering can be automated. One way to do this is below.

```

\ex[glstyle=nlevel,glhangstyle=cascade,glneveryline={\hfil\insertno,\it,},
    glnabovelineskip={,-1pt},glspace=! .4em]
\count255=114
\def\insertno{\eightrm(\the\count255)\global\advance\count255 by 1}%
\beginl[]
[Hom\^{a}o/\textsc{exist}]
[sa/one]
[\v{c}\^{o}/\textsc{clf}]
[p\^{o}/person]
[tha/old]
[\~{n}u.\textsc{3s}]
:

```

Note carefully that `\beginl` has been supplied with an empty argument. Otherwise the parser would have been confused and `[Hom...]` would have been interpreted as an optional argument. An alternative would have been `\beginl{}[Hom...]` or `\beginl\relax[Hom...]`.

14. *TeX* reserves `\count255` for scratchwork. Since the assignment needs to be global, it is important to use a count register that does not serve another purpose. If a named count register is available, it would be better to use that register.

13. Referring to examples and labeled parts of examples

13.1. Unnamed reference

Macros: `\lastx`, `\nextx`, `\blastx`, `\anextx`, `\bblastx`

If x is the value of `\excnt`, these macros produce $x - 1$, x , $x - 2$, $x + 1$, $x - 3$, respectively, expressed as an either an arabic or roman numeral, depending on the setting of `exno`. The example number counter is incremented early in the expansion of `\ex` and `\pex`, so if one of these macros is used inside an example, “last example” has the meaning “current example number”. Although `\excnt` is incremented early, it is incremented after local parameters are set. Consequently, if any of these macros is used in setting parameters in the optional argument of `\ex` or `\pex`, it acts as if it were being evaluated just before `\ex` or `\pex`.

It is potentially dangerous to use macros like `\bblastx` or `\anextx` for reference to an example because later additions or deletions in the document can throw off the reference. This kind of misreference is particularly easy to overlook in proofing a document. It is better to assign names to the things you want to refer to and to refer to them by name, particularly in a document that will undergo a lot of rewriting. If reference by name is used and an intervening example is deleted or added, no problem arises. If the example which is referred to is deleted, then *TeX* will report a missing reference.

13.2. Named reference

Macros: `\deftag`, `\deftagex`, `\deftaglabel`, `\deftagpage`, `\getref`, `\getfullref`

Parameter:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|------------------|------------------|----------------------|
| <code>tag</code> | character string | (none) |

The core macros are `\deftag` and `\getref`. `\deftag` takes two (obligatory) arguments. After `\deftag{reference}{tag}` is executed, `\getref{tag}` expands to *reference*, where the value of the reference is determined at the point the `\deftag` command was evaluated. `\deftagex{tag}` expands to `\deftag{\the\exno}{tag}`. `\deftagpage{tag}` expands to `\deftag{\the\pageno}{tag}`, but the expansion is delayed until after *TeX*'s page breaking mechanism has decided what page the `\deftag` command appears on. The tag/page pair which is generated in this case is written to a file only, so `\getref` cannot recover a page number until a tag-ref file is read.

The evaluation of `\deftaglabel{tag}` is more complex. `\pexcnt`, the setting of `labeltype`, and the setting of `everylabel` are used to construct the reference (either a letter or a number). The tag which is submitted to `\deftag` is *tag*.tag, where *tag*' is the example number tag.

The code below produces (133).

```

\pex[interpartskip=0pt]
\ a First\deftag{the first part of example \lastx}{FP}
\ a Second\deftagex{snoopy}\deftaglabel{dog}
\ a Third\deftaglabel{a}
\ xe

```

(133) a. First
b. Second
c. Third

Afterwards, assuming that the tag `snoopy` has not been redefined in the interim

```

\getref{snoopy}           → 133
\getref{snoopy.dog}      → b
\getfullref{snoopy.dog} → 133b
\getref{snoopy.a}       → c
\getfullref{snoopy.a}   → 133c
\getref{dog}            → undefined tag warning, or spurious reference
\getref{FP}             → the first part of example 133

```

`\getref{dog}` will produce a spurious reference if the tag `dog` is defined somewhere else in your document. `\deftaglabel` must know, at the point that it is expanded, what tag has been attached to the example number. If the example number has not been tagged at that point, a warning message is issued.

There are two alternatives to the explicit use of `\deftagex` and `\deftaglabel`. One is the use of the parameter `tag`. After

```

\pex[interpartskip=0pt,labeltype=alpha,tag=snoopy]
\ a First
\ a[tag=dog] Second
\ a Third\deftaglabel{A}
\ xe

```

`\getref{snoopy}`, `\getref{snoopy.dog}`, and `\getref{snoopy.A}` are interpreted as expected, as are `\getfullref{snoopy.dog}` and `\getfullref{snoopy.A}`.

The other alternative is a special “tagging notation” which, unlike most shortcuts, makes the code easier to read as well. After

```

\pex[interpartskip=0pt,labeltype=alpha]<snoopy>
\ a First
\ a<dog> Second
\ a[tag=A] Third
\ xe

```

`\getref{snoopy}`, `\getref{snoopy.dog}`, and `\getref{snoopy.A}` are interpreted as expected, as are `\getfullref{snoopy.dog}` and `\getfullref{snoopy.A}`.

If `\getref{sometag}` is evaluated and the tag is undefined, a warning message to this effect is generated and `•sometag` is typeset instead of the intended reference.

13.3. Proofing references

Macro: `\refproofing`

It is tedious to check that references to example numbers and the like are correct (i.e. refer to what you think they refer to). *ExPex* provides a little help. If you execute `\refproofing`, then all the references that are generated by `\bblastx`, `\blastx`, `\lastx`, `\nextx`, `\anextx`, `\getref`, and `\getfullref` are highlighted. If *PSTricks* is loaded at the point that `\refproofing` is evaluated, the highlighting consists of a box around the reference. If not, the reference is under- and overlined. Highlighting is helpful in copy editing. The difference is illustrated below.

- (134) a. Consider (133b), for example. default behavior
 b. Consider 133b, for example. `\refproofing`, *PSTricks* available
 c. Consider 133b, for example. `\refproofing`, *PSTricks* not available

13.4. The tag/reference file

Macros: `\gathertags`, `\tagfilesuffix`

Forward references require writing tag/reference associations to a file in one run, then reading this file in a second run. Forward reference is only one reason for such a system. If you work on a document in pieces, say one section at a time, and need to refer to things in prior sections, then the tag/reference pairs from previous sections must be stored in a file so that they can be used in the section that you are working on. If you say `\gathertags`, the tag/reference pairs will be written to a file as they are established. If your main file is named *foop.tex*, for example, the default is to write the tag/ref pairs to *foop-tags.tex*. But you can modify this by using the command `\tagfilesuffix`. *expex.tex* contains `\tagfilesuffix{-tags}`, but you can overrule this with `\tagfilesuffix{your suffix}`.

When the first `\getref`, `\getfullref`, or `\gathertags` command is encountered, *ExPex* checks to see if there is a tag file. If the file does exist, it is read and all the tag/reference pairs it encodes are established. No testing is done for conflicting tag/reference pairs with the same tag, so it is the responsibility of the user to see that this does not occur to a bad effect. If the user wants to be absolutely sure that bad references have not accidentally occurred because of using the same tag twice, he/she can look directly at the tag file in a text editor. If the lines are alphabetized, it is relatively easy to find tags for which multiple references have been established. If you are a careful copy editor, this may be worth doing in a complex project (a book length manuscript) during final copy editing. Of course, the printed final manuscript should be checked in any event to make sure that the references refer to what you want them to refer to.

For what it is worth, my own style of work is to break up projects into multiple pieces and have a main file which calls the various pieces. Then I simply comment out the calls to pieces that I am not working on. From time to time I will run the main file calling on all the various pieces, invoking `\gathertags`. Then I comment out `\gathertags`. The tag-ref file which is created then remains intact until `\gathertags` is invoked at some future time. All of the tag/reference pairs it encodes are available in subsequent work until a new tags file is created.

13.5. References to references, references as values

Macro: `\lastlabel`

If, for some reason, you want to refer to a specific part of a multipart example without tagging the example number, it can be done as follows, assuming that the part labels are alphabetical.

```
\pex[everylabel=\it]
\ a First Example.
\ a Second Example.\deftag{\lastx\lastlabel}{snoopy}
\ a Third Example.
\ xe
```

(135) *a.* First Example.
b. Second Example.
c. Third Example.

You can then use `\getref{snoopy}` to retrieve 135*b*. `\lastlabel` expands to the most recent label, in the form in which it is printed (i.e. containing the expansion of `\everylabel`). If the part labels are numerical, then `\deftag{\lastx.\lastlabel}` is required.

You can also say:

```
\ex[exno=\getref{snoopy}] Second example.\ xe
```

(135*b*) Second example.

Or you can say:

```
\ex[exno={\getref{snoopy}, repeated}, exnoformat={X}]
Second example.\ xe
```

[135*b*, repeated] Second example.

Braces must hide the comma in the value which sets the key.

If you say

```

\pex[labeltype=numeric]<dog>
\a First Example.
\a<G> Second Example.
\a Third Example.
\xe

```

```

(136) 1. First Example.
      2. Second Example.
      3. Third Example.

```

then, if you want to repeat (136.2) at some point, you can use:

```

\ex[exno=\getfullref{dog.G}] Second example\xe

```

```

(136.2) Second example

```

13.6. Extensions of the tag/reference mechanism

The tag/reference mechanism can easily be extended to reference to chapters, sections, and subsections. Suppose, for example, that there are counters for the chapter number, section number, subsection number, and subsubsection number. One might define:

```

\def\currsec{\the\chapterno
  \ifnum\secno>0 .\the\secno
  \ifnum\subsecno>0 .\the\subsecno
  \ifnum\subsubsecno>0 .\the\subsubsecno \fi\fi\fi}
\def\deftagsec#1{\deftag\currsec{#1}}

```

This assumes that there are counters `\chapterno`, `\secno`, `\subsecno`, and `\subsubsecno` and that when a chapter is initiated, `\secno` is set to 0, when a section is initiated, `\subsecno` is set to 0, and that when a subsection is initiated, `\subsubsecno` is set to 0. (Lines 1–5, commented out, are included at the end of *expex.tex* for the user to use, modify, or ignore.)

13.7. The parameter `fullreformat`

Parameter:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|---------------------------|------------------------------|----------------------|
| <code>fullreformat</code> | <code>... X ... A ...</code> | <code>XA</code> |

1. `labelformat` (see page 19) determines how the labels are formatted. Generally they are followed by a period, but this parameter allows other possibilities.

2. `fullreformat` determines how references retrieved by `\getfullref` are formatted. Generally, example numbers and labels are concatenated if the labels are characters and separated by a period if the labels are numbers. This parameter allows other possibilities.

The format specification is primitive. Val(fullreformat) is analyzed as #1X#2A#3, with X preceding A. Then the full reference is found by substituting the example number for X and the part label for A.

Consider the artificial example below, with unusual formatting for illustrative purposes.

```
\pex[labeltype=numeric,labelformat={A}],
  fullreformat=X-A,samplelabel={{1}}<K>
\ a first
\ a<A> second
\ xe
```

| | | |
|-------|-----|--------|
| (137) | [1] | first |
| | [2] | second |

\getref{K.A} produces 2 and \getfullref{K.A} produces 137-2.

13.8. Reference to a part of a multipart example

Suppose there is a multipart example which you want to partially repeat at a later point, giving only some of the parts. There are many opportunities to make a mistake. The reference to the example number may be wrong, the references to the part labels may be wrong, or a particular part may not be repeated the way it originally appeared. Here is one way to (partially) automate the references to avoid most of the possible reference errors.

```
\deftag{Someone loves everyone.}{1st}
\deftag{Everyone loves someone.}{2nd}
```

```
\pex<A>
\ a \dots
\ a<X> \getref{1st}
\ a \dots
\ a<Z> \getref{2nd}
\ xe
```

```
\ex An intervening example.\xe
```

The two crucial examples in (\getref{A}) are repeated below:

```
\pex[exno=\getref{A}]
\ a[label=\getref{A.X}] \getref{1st}
\ a[label=\getref{A.Z}] \getref{2nd}
\ xe
```

```
\ex Numbering resumes.\xe
```

| | |
|-------|----------------------------|
| (138) | a. ... |
| | b. Someone loves everyone. |
| | c. ... |
| | d. Everyone loves someone. |

(139) An intervening example.

The two crucial examples in (138) are repeated below:

| | |
|-------|----------------------------|
| (138) | b. Someone loves everyone. |
| | d. Everyone loves someone. |

(140) Numbering resumes.

Important limitation: Inside an example that sets exno to a nonempty value, the use of \deftaglabel or \deftagex, or their implicit versions using the <...> notation, or \label, will result in an error or (more dangerously) an unexpected outcome. This is a minor limitation because there should be no need to tag items which themselves are referenced by tags. It is important only because violating it can generate Tex errors.

13.9. Support for the LaTeX `\label` and `\ref` commands

LaTeX users, if so inclined, may wish to use the LaTeX `\label`-`\ref` mechanism to tag example numbers and example part labels.

Assuming that LaTeX is in use,

```
\pex \label{A}
\ a Tom\label{B}
\ a Dick\label{C}
\ a Harry\label{D}, etc.
\xe
```

```
\ex~ Intervening example.\xe
```

```
\noindent Consider example (\ref{A}).
Tom is in example (\ref{B}), Dick is
in example (\ref{C}), and Harry is in
example (\ref{D}).
```

(141) a. Tom
b. Dick
c. Harry, etc.

(142) Intervening example

Consider example (141). Tom is in example (141a), Dick is in example (141b), and Harry is in example (141c).

There are important differences between the LaTeX mechanism and the `\deftag`-`\getref` mechanism.

1. `\ref{tag}` cannot be used as a special example number, nor can it be given as an optional argument to `\a`.
2. The `\label` name space is flat. No distinction is made between names of examples and names of parts of examples. This makes it much harder to remember and manage the names, at least for me.
3. LaTeX provides no control over the generation of the file containing the tag-reference pairs. One cannot work on Chapter 5 (for example) and have all the tag-reference information for the earlier chapters available without actually submitting all of the Chapters 1 to 4 to LaTeX while you are working on Chapter 5.

14. Tables in examples

Macro: `\hwit`

Most of the difficulty of formatting example displays which contain tables comes from formatting the table itself. This manual will not teach the reader how to use *TeX* to format tables. It will be assumed that the reader knows how to use the *TeX* primitive `\halign` or the *LaTeX* macros based on `\halign`. But this section might contribute something to understanding how to use the table making tools in linguistics examples.

Many tabular examples have the form

```
\ex \vtop{\halign{ ... }}\xe
```

For example:

| | | | |
|-------|-------|----------|------------|
| (143) | baudh | bu-baudh | know, wake |
| | smai | si-smai | smile |
| | suap | su-suap | sleep |
| | miaks | mi-miaks | glitter |
| | auc | u-auc | please |

```
\ex \vtop{\halign{%  
#\hfil&& \quad #\hfil\cr  
baudh& bu-baudh& know, wake\cr  
smai& si-smai& smile\cr  
suap& su-suap& sleep\cr  
miaks& mi-miaks& glitter\cr  
auc& u-auc& please\cr  
}}\xe
```

A more elaborate version of (143), with a title and labeled columns, is given below. `\hwit` is described below.

| | | | |
|-------|--|---------------------|--------------|
| (144) | The perfect stems of some roots with a high vowel in their nucleus | | |
| | <i>root</i> | <i>perfect stem</i> | <i>gloss</i> |
| | baudh | bu-baudh | ‘know, wake’ |
| | smai | si-smai | ‘smile’ |
| | suap | su-suap | ‘sleep’ |
| | miaks | mi-miaks | ‘glitter’ |
| | auc | u-auc | ‘please’ |

```
\ex The perfect stems of some roots with a  
high vowel in their nucleus\par\nobreak\medskip  
\quad\vbox{\halign{%  
#\hfil&& \hskip3em #\hfil\cr  
\hfil\hwit{root}& \hfil\hwit{perfect stem}&  
 \hfil\hwit{gloss}\cr  
\noalign{\smallskip}  
baudh& bu-baudh& ‘know, wake’\cr  
smai& si-smai& ‘smile’\cr  
suap& su-suap& ‘sleep’\cr  
miaks& mi-miaks& ‘glitter’\cr  
auc& u-auc& ‘please’\cr  
}}\xe
```

`\hwt` (hidewidth italics) inserts the italicized label into the alignment in such a way that it is centered over the nonwhite portion of the column it heads. It hangs over equally on both sides if necessary. Hiding the width of column labels is often important so that the column labels do not affect the column widths.

`\par\nobreak` appears after the title so that page breaking does not detach the title from the table that follows.

14.1. Tables with labeled lines

Macros: `\labels[]`, `\tl`, `\nl`

If reference must be made to particular lines in (144), the lines need labels of some sort. One approach is to explicitly enter the line labels a–e:

(145) The perfect stems of some roots with a high vowel in their nucleus

| | <i>root</i> | <i>perfect stem</i> | <i>gloss</i> |
|----|-------------|---------------------|--------------|
| a. | baudh | bu-baudh | ‘know, wake’ |
| b. | smai | si-smai | ‘smile’ |
| c. | suap | su-suap | ‘sleep’ |
| d. | miaks | mi-miaks | ‘glitter’ |
| e. | auc | u-auc | ‘please’ |

```
\ex The perfect stems of some roots with a
high vowel in their nucleus\par\nobreak\medskip
\quad\ vbox{\halign{%
#\hfil& \quad #\hfil&& \hskip3em #\hfil\cr
& \hfil\hwt{root}& \hfil\hwt{perfect stem}&
\hfil\hwt{gloss}\cr
\noalign{\smallskip}
a.& baudh& bu-baudh& ‘know, wake’\cr
b.& smai& si-smai& ‘smile’\cr
c.& suap& su-suap& ‘sleep’\cr
d.& miaks& mi-miaks& ‘glitter’\cr
e.& auc& u-auc& ‘please’\cr
}}\xe
```

If a line is deleted or added, or if lines are interchanged for some reason, considerable relabeling may be required.

ExPex provides some macros which simplify this code and make it easier to manipulate. They are used in the alternate code for (145) below and described below. `\labels` initializes the counter `\pexcnt`, which is then used to generate the labels. It also activates the macros `\tl` (table label) and `\nl` (no label). `\tl` inserts the appropriate label, with following period, and increments the counter. `\nl` abbreviates `\omit\hfil`, so that it can be used to prevent the appearance of a label in a cell. `\labels` takes parameters, so you can say things like

`\labels[labeltype=caps, everylabel=\it]`. Of course, these parameters can also be set at the `\ex` level, if desired, or even globally.

```
\ex The perfect stems of some roots with a
high vowel in their nucleus\par\nobreak\medskip
\quad\ vbox{\labels\halign{%
\tl #\hfil& #\hfil& \quad #\hfil&& \hskip3em #\hfil\cr
\nl & \hfil\hwit{root}& \hfil\hwit{perfect stem}&
\hfil\hwit{gloss}\cr
\noalign{\smallskip}
& baudh& bu-baudh& 'know, wake'\cr
& smai& si-smai& 'smile'\cr
& suap& su-suap& 'sleep'\cr
& miaks& mi-miaks& 'glitter'\cr
& auc& u-auc& 'please'\cr
}}\xe
```

The advantages of implicit line label insertion should be obvious. One often decides to insert another entry, or to delete an entry. If the labels are inserted explicitly, this usually requires changing multiple labels. If the table has many lines, this is particularly onerous. References in the text to particular lines also need to be changed to match the new labeling. We will see below that lines in tables can be named and reference made by name, using implicit line numbering.

14.2. Tagging implicit labels in tables

`\deftaglabel` can be used to associate a tag with a label that is introduced into a table by `\tl` (“table label”). `\tl` can also read an optional tag using the `<...>` mechanism.

```
\ex<Washo>
\vtop{\labels\halign{\tl #\hfil&& \quad #\hfil\cr
\nl & \hwit{Root}& \hwit{Plural}& \hwit{Gloss}\cr
& baloxat& baloxaxat& bows\cr
& moya& moyaya& shoulder\cr
<A>& nent'us& net'unt'us& old women\cr
<B>& mokgo& mogokgo& shoes\cr
}}
\xe
```

Examples (`\getfullref{Washo.A}`) and (`\getfullref{Washo.B}`) are the most complex, and therefore the most revealing. Examples (`\getref{Washo}\getref{Washo.A}`, `\getref{Washo.B}`) are the most complex, and therefore the most revealing.

| (146) | <i>Root</i> | <i>Plural</i> | <i>Gloss</i> |
|-------|-------------|---------------|--------------|
| a. | baloxat | baloxaxat | bows |
| b. | moya | moyaya | shoulder |
| c. | nent'us | net'unt'us | old women |
| d. | mokgo | mogokgo | shoes |

Examples (146c) and (146d) are the most complex, and therefore the most revealing. Examples (146c,d) are the most complex, and therefore the most revealing.

14.3. Some useful table making tools

Macro: `\tspace[]`, `\crs`

Parameters:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|--------------------------|--------------|----------------------|
| <code>dima†</code> | dimension | 2.4em |
| <code>dimb†</code> | dimension | (not set) |
| <code>dimc†</code> | dimension | (not set) |
| <code>crskip\user</code> | skip | 1.2ex |

Tables often need considerable adjustment in order to balance the needs of readability, space limitations, and matching the typographic structure to the conceptual structure. Sometimes this requires a delicate balancing act. *ExPex* provides three parameters (scratch dimensions) `dima`, `dimb`, and `dimc`, and corresponding macros which expand to their settings, which can be used for this. Additionally, `\tspace[key]` expands to `\hskip\lingtag`, so that horizontal skip like `\tspace[dima]` or `\tspace[textoffset]` can be easily used in table construction. The code for (146) could be written:

```
\ex[textoffset=1em,dima=1em,dimb=3em]
The perfect stems of some roots with a high vowel in their
nucleus\par\nobreak\medskip
\tspace[dima]\vbox{\labels\halign{%
\tl #\hfil& \tspace[textoffset]\# \hfil&& \tspace[dimb]\# \hfil\cr
\nl & \hfil\hwit{root}& \hfil\hwit{perfect stem}&
 \hfil\hwit{gloss}\cr
:
}
```

Localizing all the parameters which might need adjustment in the optional argument of `\ex` helps organize the adjustment process.

If no optional argument is supplied to `\tspace`, it expands to `\hskip\lingdima`.

The macro `\crs` and command key `crskip` are provided to assist in fine tuning the vertical spacing inside an alignment. `\lingcrskip` expands to the setting of `crskip`, and

```
\crs → \cr \noalign{\vskip\lingcrskip}
```

14.4. Tables that can break between pages

Macros: `\exdisplay[]`, `\noexno`, `\exnoprint`, `\crnb`

Up to this point, only tables that are typeset in a `vbox` have been considered. The *TeX* page breaking algorithm does not split a box between pages. Sometimes, an especially tall table is needed and one has the choice of floating the table to the top of the next page (using *TeX*'s `\topinsert`) or constructing a breakable table. The latter choice is often preferable and can be implemented using *TeX*'s primitive `\halign`.

The example number must be part of the `\halign`, not inserted by `\ex`. `\exdisplay` is designed to accommodate an unboxed `\halign`. Like `\ex`, `\exdisplay` must be closed by `\xe`. `\exdisplay ... \xe` is just like `\ex ... \xe` except that an example number is not printed and the horizontal dimensions `\numoffset` and `\textoffset` are irrelevant. `aboveexskip` and `belowexskip` play the same role, paragraph indentation is cancelled, `\lingeveryex` and `\lingEveryex` are executed, and `\excnt` is advanced.

`\noexno` is provided in case the user wants to use `\exdisplay` for something other than a numbered example. It cancels the automatic advancement of `\excnt`. You can say `\exdisplay\noexno` or `\exdisplay[...]\noexno` to cancel `\excnt` advancement.

Now consider the ‘tall display’ (147). It is sufficiently tall that it either must be put in an insertion or typeset so that it can be split by a page break.

(147) High vowel in the nucleus of the root

| | <i>root</i> | <i>perfect stem</i> | <i>gloss</i> |
|----|-------------|---------------------|--------------|
| a. | baudh | bu -baudh | ‘know, wake’ |
| b. | stau | tu -stau | ‘praise’ |
| c. | smai | si -smai | ‘smile’ |
| d. | suap | su -suap | ‘sleep’ |
| e. | miaks | mi -miaks | ‘glitter’ |
| f. | auc | u -auc | ‘please’ |

No high vowel in the nucleus of the root

| | | | |
|----|-------|------------------|--------------|
| g. | suaj | sa -suaj | ‘embrace’ |
| h. | krand | ka -krand | ‘cry out’ |
| i. | skand | ka -skand | ‘leap’ |
| j. | mard | ma -mard | ‘rub, crush’ |
| k. | mnaa | ma -mnaa | ‘note’ |

The example number in the code below is inserted by `\exnoprint`, which is how `\ex` inserts example numbers. If, by the way, you don’t like the way example numbers are inserted (surrounded by parentheses), you can redefine `\exnoprint`. The code uses `\crnb`, which expands to `\cr\noalign{\par\nobreak}`, to prevent a page break between a heading and the remainder of the table which follows. Page breaks are encouraged by `\exbreak` at a few points where it is judged that the logic of the table can tolerate it. `\exbreak` encourages a page break at the

cost of allowing a certain amount of white space at the bottom of the broken page.¹⁵ A negative horizontal skip `\tspace[dimc]` is used to highlight the subheadings (and to show the reader that it is a possibility, if you need it).

```
\exdisplay[labeloffset=2em,dima=2em,dimb=1em,dimc=-.8em]
\def\#1-{\bf #1}-}%
\labels
\openup1pt
\halign{%
  #\tspace[labeloffset]\hfil&    % example number
  #\tl\tspace[textoffset]\hfil&  % line label
  #\tspace \hfil&                % root
  #\tspace \hfil&                % perfect stem
  ‘#’\hfil\cr
(\the\excnt)& \omit\tspace[dimc] High vowel in the nucleus of the
  root\hidewidth\crnb
& \nl & \hwit{root}& \hwit{perfect stem}&
  \omit\tspace\hwit{gloss}\crnb
&& baudh& \bu-baudh& know, wake\cr
&& stau& \tu-stau& praise\cr
&& smai& \si-smai& smile\cr
\noalign{\exbreak}
&& suap& \su-suap& sleep\cr
&& miaks& \mi-miaks& glitter\cr
&& auc& \u-auc& please\cr
\noalign{\exbreak\smallskip}
& \omit\tspace[dimc] No high vowel in the nucleus of the
  root\hidewidth\crnb
&& suaj& \sa-suaj& embrace\cr
&& krand& \ka-krand& cry out\cr
\noalign{\exbreak}
&& skand& \ka-skand& leap\cr
&& mard& \ma-mard& rub, crush\cr
&& mnaa& \ma-mnaa& note\cr
}\xe
```

15. See page 76 for a more complete discussion of `\exbreak`. The amount of encouragement and the amount of whitespace are parametrized.

14.5. Squeezing tables into tight places

Foregoing `\ex` and directly using `\halign` inside `\exdisplay ... \xe` has other uses besides constructing tables in numbered examples which can be broken between pages. The technique is also sometimes useful in fitting a table in a numbered example into a narrow page width. The table below was constructed to fit on a page of width 4.3 in, with no room to spare. It gives the present indicative conjunction of the Sanskrit verb root *dveṣ/dviṣ* ‘hate’.

| | | | | | | |
|-------|---------------------------|-------------------|-------------------|-----------------|-------------------|-------------------|
| (148) | <i>Present Indicative</i> | | | | | |
| | <i>active</i> | | | <i>middle</i> | | |
| | <i>sg</i> | <i>du</i> | <i>pl</i> | <i>sg</i> | <i>du</i> | <i>pl</i> |
| 1 | dvéṣ -mi | dviṣ- vás | dviṣ- más | dviṣ- é | dviṣ- váhe | dviṣ- máhe |
| 2 | dvék -ṣi | dviṣ- ṭhás | dviṣ- ṭhá | dvikṣ- é | dviṣ- áthe | dviḍ- ḍhvé |
| 3 | dvéṣ -ṭi | dviṣ- ṭás | dviṣ- ánti | dviṣ- ṭé | dviṣ- áte | dviṣ- áte |

Assuming that the page width has been set to 4.3 in (`\hsize=4.3in`), the following code produces (148), which is precisely 4.3 in wide.

```
\exdisplay[dima=.5em,dimb=.4em,textoffset=.5em]
\def\#1{${\acute{\hbox{\=#1}}}$}%
\tabskip=0pt
\openup.4ex
\halign to \hsize{\tspace[dima]#\tspace[textoffset]\hfil&
  #\hfil\tabskip=0pt plus 1fil&
  #\hfil& #\hfil& \tspace[dimb]#\hfil&
  #\hfil & #\hfil\tabskip=0pt\cr
\omit\exnoprint\hidewidth&
  \multispan6 \hwit{Present Indicative}\crnb
&\multispan3 \hwit{active}& \multispan3 \hwit{middle}\cr
& \hwit{sg}& \hwit{du}& \hwit{pl}&
  \hwit{sg}& \hwit{du}& \hwit{pl}\cr
\it 1& {\bf dv\'e\.s}-mi& dvi\.s-v\'as& dvi\.s-m\'as&
  dvi\.s-\'e& dvi\.s-v\'ahe& dvi\.s-m\'ahe\cr
\it 2& {\bf dv\'ek}-.si& dvi\.s-.th\'as& dvi\.s-.th\'a&
  dvik\.s-\'e& dvi\.s-\\athe& dvi\.d-.dhv\'e\cr
\it 3& {\bf dv\'e\.s}-.ti& dvi\.s-.t\'as& dvi\.s-\'anti&
  dvi\.s-.t\'e& dvi\.s-\\ate& dvi\.s-\'ate\cr
}
\xe
```

15. ExPex and PSTricks

Macro: `\Lingset`

Several features of *ExPex*, listed in (149), come into play only if *pstricks.tex* has been loaded at the point that *expex.tex* is loaded. They are intended to make it easier to use *PSTricks* in examples.

- (149)
1. `\Lingset` is activated. It works like `\lingset`, but if there are parameters which are not in the family *ling*, they are passed to `\psset`.
 2. The family *ling* is added to the set of parameter families which `\psset` can set.
 3. The optional argument of `\ex`, `\pex`, and `\exdisplay` is passed to `\Lingset`.

`\Lingset` first scans its argument from left to right and sets all the parameters from the family *ling*. The remaining keys are then passed to `\psset`. If *PSTricks* has not been loaded, `\Lingset` is defined, but its meaning is the meaning of `\lingset`.

This is illustrated by (150).

(150) Whom did John persuade t [PRO to visit whom]



(The example is from Juan Uriagereka, in *The Role of Economy Principles in Linguistic Theory*.)

In the code below, note that the optional argument of `\ex` is used to set the *PSTricks* parameters `angle`, `arrows`, `nodesep`, `labelsep`, and `lineararc`, and the *ExPex* parameter `dima`. Note also that the *ExPex* scratch dimension `\lingdima` is used seamlessly by the *PST-Node* macros.

```

1 \ex[angle=-90,nodesep=0pt,arrows=->,dima=.2em, labelsep=.25ex,
2   lineararc=.7ex]
3 \def\#1(#2){\rnode{#2}{\strut #1}}%
4 %
5 \vrule height0pt depth5.3ex width0pt
6 \\\Whom(A) did John persuade \\\t(B) [ PRO to visit \\\whom(C) ]
7 \ncbar[armA=3.5ex,offsetB=\lingdima]{B}{A}
8 \bput{0}{\$M_{sp}=2\$}
9 \ncbar[armA=4.5ex,offsetB=-\lingdima]{C}{A}
10 \bput{0}{(1.2){\$M_{sp}=6\$}}
11 \xe

```

Since the connections which *PSTricks* draws are dimensionless, a zero width `\vrule` is used to give correct spacing. (The width can first be made nonzero so it is visible, and the depth adjusted.)

The following gives the same result.

```

1 \ex
2 \psset{angle=-90,nodesep=0pt,arrows=->,dima=.2em,labelsep=.25ex,
3   linearc=.7ex}
   :
```

`\Lingset` and `\psset` are not entirely equivalent, even when *PSTricks* is active. If a key name in the family *ling* is the same as a key name in another family which has also been added as part of the *PSTricks* extended family, `\Lingset` will treat it unambiguously as in the *ling* family. `\psset`, on the other hand, does not give priority to keys in the *ling* family. Exactly how `\psset` establishes priority is a complex matter, depending on the history of the formation of the *PSTricks* extended family.

16. Control over page breaking inside examples

Page breaking is generally much more disruptive if it occurs inside an example than if it occurs in a paragraph of text. Steps can be taken to minimize the possibility that examples are disrupted by page breaks. Avoiding disruption can mean either that a page break does not occur, or if it does, it happens at a “good place”. If the parts of a multi-part example naturally form subgroups, it is desirable to encourage a break between subgroups and discourage a break inside a subgroup.

ExPex attempts to provide tools to help avoid breaking examples between pages and, if they must be split, making it easy to specify where in the example splitting should be encouraged or discouraged. This must be balanced with limitations on how much white space at the bottom of pages is allowable.

The amount of whitespace at the bottom of pages and how much variability in the vertical skip between examples and the surrounding text is acceptable will probably vary between different intended purposes: handout, draft, paper, camera-ready paper, lecture notes, camera-ready book manuscript, etc. It will also vary between different stages of production. Different editors may have different views of the tradeoff between keeping examples intact and allowing whitespace at the bottom of pages. There is no sense in spending time on formatting while you are still working on a draft. *ExPex* therefore provides a number of parameters which can be adjusted to control various aspects of page breaking.

In spite of efforts to guide *TeX* in making good decisions about page breaking, the results are not always satisfactory.

Every once and a while, *TeX* will produce a really awful looking page and you will wonder what happened. For example, you might get just one paragraph and a whole lot of white space, when some of the text on the following page would easily fit into the white space. The reason for such apparently anomalous behavior is almost always that no good page break is possible; even the alternative that looks better to you is quite terrible as far as *TeX* is concerned!

(from Knuth’s *TeXbook*, p. 115)

In situations like this, one has to give up on encouraging *TeX* to do the right thing and give explicit orders to break the page by inserting `\eject` or `\vfil\eject` in the appropriate place. The drawback to an explicit order is that it remains in force after revisions are made to the document. If it causes page breaking that is obviously bad, the problem will usually be seen when editing. The biggest danger is that the output is not terrible, but it is less than ideal and the discrepancy is missed in less than meticulous copy editing.

16.1. Discouraging page breaks in examples

Macro: `\exbreak`

Parameters:

| <i>key</i> | <i>value</i> | <i>initial value</i> |
|--------------------------------|--------------|----------------------|
| <code>exbreakfil</code> | skip | 0pt plus 4ex |
| <code>exbreakpenalty</code> | integer | -500 |
| <code>splitpartspenalty</code> | integer | 200 |

TeX carries out page breaking by assigning a cost (via penalties) to each possible page break and choosing the lowest cost option. Plain *TeX* defines a macro `\goodbreak` which ends the current paragraph and gives a negative penalty (-500) if a page break is taken after the paragraph. A negative penalty is a reward, which encourages page breaking at that point. The *ExPex* macro `\exbreak` is a variant of `\goodbreak`. It not only ends the current paragraph and awards a penalty (a reward if the penalty is negative) if a page break is taken at that point, but makes it easier for a satisfactory page break to be taken by allowing the page to end with some vertical fill, allowing some whitespace to appear at the bottom of the broken page.

`\exbreak` is inserted at the beginning of every `\ex` or `\pex` block. The penalty and amount of fill are parameterized. The default values are suitable for camera-ready copy, assuming that the publisher is willing to tolerate a small amount of whitespace at the bottom of a page in return for keeping an example intact. 4 ex is about 1.5 times the distance between baselines in most fonts. For drafts, I set `exbreakfil=.3\vsiz`, which keeps almost all examples intact at the cost of allowing substantial whitespace to appear at the bottom of a page.

The setting of `exbreakfil` can be overridden by supplying a value directly as an optional argument of `\exbreak`. For example, `\exbreak[5ex]` acts like `\exbreak` (with no argument) with `exbreakfil` is set to 5ex.

Page breaks right before an example part in a `\pex` construction introduced by `\a` are discouraged by imposing a penalty determined by the setting of `splitpartspenalty`. If `splitpartspenalty` is set to 10000, such breaks will be completely disallowed. I use this setting for writing drafts. Coupled with a generous setting of `\exbreakfil`, almost all page breaks in examples are eliminated, which is what I prefer in writing a draft. The default value is more suitable for finished documents.

In older versions of *ExPex*, `\exbreak` was called `\goodpar`. The present name is more consistent with *TeX*'s naming habits.

16.2. Controlling where page breaks occur in examples (if they must)

If a page break within an example cannot be avoided because it would require too much vertical fill to be inserted, it is often important to control where within the example the page break is made. Consider for example a display like (151).

- (151) a. example A
b. contrast with example A
c. example B
d. variation on B
e. another variation on B
f. a third variation on B
g. example C
h. contrast with example C

Suppose the logic of the examples pairs a and b, c–f, and g and h. How do you avoid a page break which interrupts the logic of the collection of examples? One option is the standard *TeX* method of using `\nobreak` to prevent a page break, as in the code below:

```
\pex[interpartskip=.25ex]
\ a example A\par\nobreak
\ a contrast with example A
\ a example B\par\nobreak
\ a variation on B\par\nobreak
\ a another variation on B\par\nobreak
\ a a third variation on B
\ a example C\par\nobreak
\ a contrast with example C
\ xe
```

Note that `\par` must precede `\nobreak`. We are interested in preventing a page break in the process of adding lines to the current page, not in preventing a break in the process of building the lines which are added to the current page. `\par` breaks the line, which takes *TeX* out of line building mode and puts it into page building mode.

`\exbreak` offers an alternative.

```
\pex[interpartskip=.25ex]
\ a example A
\ a contrast with example A\exbreak
\ a example B
\ a variation on B
\ a another variation on B
\ a a third variation on B\exbreak
\ a example C
\ a contrast with example C
\ xe
```

Index of control sequences, parameters, and special symbols

`*`, parameter 32
`+`, exceptional gloss item 39
`+`, gloss diacritic 45
`@`, exceptional gloss item 40
`@`, gloss diacritic 45
`~`, tilde diacritic 12
`\a` 15
`aboveexskip†` 10
`aboveglbskip†` 38
`aboveglcskip†` 38
`aboveglftskip†` 38
`\actualexno` 26
`\anextx` 60
`appendtopexarg` 26
`avoidnumlabelclash` 26
`\blastx` 60
`\beginl[]` 34
`\beginlpanel[]` 53
`belowexskip†` 10
`belowglpreambleskip†` 38
`belowgpreambleskip†` 15
`\blastx` 60
`\crnb` 71
`\crs` 70
`crskip\user` 70
`\defineglwlevels` 51
`\definelabeltype` 23
`\definelingstyle` 30
`\deftag` 60
`\deftagex` 60
`\deftaglabel` 60
`\deftaglabel, in tables` 69
`\deftagpage` 60
`dima†` 70
`dimb†` 70
`dimc†` 70
`\endgl` 34
`\endpanel` 53
`\endpreamble` 42
`Everyex†` 30
`everyex†` 30
`everygl` 37
`everygla` 37
`everyglb` 37
`everyglc` 37
`everyglft` 37
`everyglilg` 37
`everyglpreamble` 37
`everygluf` 56
`everylabel` 19
`everypanel†` 53
`everytrailingcitation` 48
`\ex` 10
`\exbreak` 76
`exbreakfil` 76
`exbreakpenalty` 76
`\excnt` 10
`\exdisplay` 71
`exno` 13
`exno, setting by reference` 63
`exnoformat` 13
`\exnoprint` 71
`exnotype` 13
`exskip` 10
`extraglskip†` 38
`fullrefformat` 64
`\gathertags` 62
`\getfullref` 60
`\getref` 60
`\gla[]` 34
`\glb[]` 34
`\glc[]` 34
`\glft[]` 34
`glftpos` 52
`glhangindent` 38
`glhangstyle` 38
`glnabovelineskip` 44
`glneveryline` 44
`\glpreamble[]` 34
`glrightskip` 38, 50
`glspace†` 38
`glstruts (footnote)` 37
`glstyle` 42
`\gluf` 56

| | | | |
|--|--------|---|--------|
| <code>glufcloseup</code> | 56 | <code>\nogloss</code> (exceptional gloss token) | 39 |
| <code>glwidth</code> | 47 | <code>nopreamble</code> | 17 |
| <code>glwordalign</code> | 58 | <code>numoffset†</code> | 10 |
| <code>\hwit</code> | 66 | <code>\pex</code> | 15 |
| <code>interpartskip†</code> | 15 | <code>\pexcnt</code> | 15, 23 |
| <code>\judge</code> | 32 | <code>pexcnt</code> (parameter) | 23 |
| <code>\keepexcntlocal</code> | 28 | <code>preambleanchor</code> | 18 |
| <code>label</code> | 18 | <code>preambleoffset†</code> | 15 |
| <code>\label</code> , use of the LaTeX macro | 66 | <code>\ref</code> , use of the LaTeX macro | 66 |
| <code>labelalign</code> | 21 | <code>\refproofing</code> | 62 |
| <code>labelanchor</code> | 18 | <code>\rightcomment</code> | 48 |
| <code>labelformat</code> | 19 | <code>sampleexno</code> | 25 |
| <code>labelgen</code> | 23 | <code>samplelabel</code> | 15 |
| <code>labellist</code> | 23 | <code>splitpartspenalty</code> | 76 |
| <code>labeloffset†</code> | 15 | <code>ssratio</code> | 52 |
| <code>\labels</code> | 68 | <code>ssrightskip</code> | 52 |
| <code>labeltype</code> | 15, 23 | <code>sssep</code> | 52 |
| <code>labelwidth†</code> | 15 | <code>tag</code> | 60 |
| <code>\lastlabel</code> | 63 | <code>\tagfilesuffix</code> | 62 |
| <code>\lastx</code> | 60 | <code>textanchor</code> | 18 |
| <code>\Lingset</code> | 74 | <code>textoffset†</code> | 10 |
| <code>\lingset</code> | 8 | <code>\tl</code> | 68 |
| <code>lingstyle</code> | 30, 31 | <code>\trailingcitation</code> | 48 |
| <code>\ljudge</code> | 32 | <code>\tspace</code> | 70 |
| <code>mincitesep</code> | 48 | <code>\xe</code> | 10 |
| <code>\nextx</code> | 60 | | |
| <code>\nl</code> | 68 | | |
| <code>\noexno</code> | 71 | | |