ExPex

for linguists

Example formatting and reference

(1) a.

Mary$_i$	*ist*	*sicher,*	*dass*	*es*	*den*	*Hans*	*nicht*	*stören*	*würde*
Mary	is	sure	that	it	the-ACC	Hans	not	annoy	would

seiner	*Freundin*	*ihr*$_i$	*Herz*	*auszuschütten.*
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*
Mary	is	sure	that	his-DAT	girlfriend-DAT	her-ACC	heart-ACC

auszuchütten	*dem*	*Hans*	*nicht*	*schaden*	*würde.*
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

December 2007
Version 1.0 (Beta)

Contents

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. *ExPex* 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 *ExPex* is not based on *PSTricks*, it does handle parametrization in 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 invitably 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 *ExPex* 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 they can be fixed before the macro package leaves its beta stage.

This User's Guide begins with four examples which demonstrate *ExPex* 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 *ExPex* 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 *ExPex* solely on the basis of the four demo pages and learn the more subtle capabilites 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.

2. *ExPex* in action

1. Examples and examples with parts

```
Example (\nextx) is well-known from the literature on parasitic
gaps.  Here we are concerned with example formating, 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 straighforward.

\pex Two examples of parasitic gaps.
\a He is the man that John did not interview {\sl e\/} before
he gave the job to {\sl e}.
\a 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 (33) is well-known from the literature on parasitic gaps. Here we are concerned with example formating, not with the interesting syntax.

(33) 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 straighforward.

(34) 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 (34b), perhaps John knew that the “someone” went to prep school with the owner of the business.

---

2. Referring to examples and parts of examples by name

```
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<X5> 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.X5} of example (\getref{pg}), or (\getfullref{pg.X5})
or part \getref{pg.X5} of example (\getref{pg}).  If you use a
tag that has not been defined, {\sl ExPex\/} will let you know.
If you try to reference \getref{pg.X3}, for example, a warning
will be issued and the tag typeset.  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.

(35) 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 (35), or (35b) or part b of example (35). If you use a tag that has not been defined, *ExPex* will let you know. If you try to reference •`pg.X3`, for example, a warning will be issued and the tag typeset. 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

```
First, a simple gloss.  The hard part is remembering to leave a
space before the slash which terminates the a and b lines.

\ex
\begingl
\gla Ich habe es ihm gestern gesagt. /
\glb I have it {to him} yesterday told. /
\glc {‘I told it to him yesterday.’}
\endgl
\xe

In a gloss, if the source or gloss needs more than one line, the
user has to decide where to break the line.

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

First, a simple gloss. The hard part is remembering to leave a space before the slash which terminates the a and b lines.

(36)

Ich	*habe*	*es*	*ihm*	*gestern*	*gesagt.*
I	have	it	to him	yesterday	told.

‘I told it to him yesterday.’

In a gloss, if the source or gloss needs more than one line, the user has to decide where to break the line.

(37)

Mary$_i$	*ist*	*sicher,*	*dass*	*es*	*den*	*Hans*	*nicht*	*stören*	*würde*
Mary	is	sure	that	it	the-ACC	Hans	not	annoy	would

seiner	*Freundin*	*ihr*$_i$	*Herz*	*auszuschütten.*
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.’

---

4. *Parameters*

```
The following illustrates how parameter settings can affect the
display.  Compare (\nextx) with (\lastx) on the previous page.

\ex[everygla=\rm,everyglc=\it,aboveglcskip=1.5ex,
   moregloffset=1em,abovemoreglskip=.5ex,glspace=1.5em]
\begingl
\gla Mary$_i$ ist sicher, dass es den Hans nicht st\"oren w\"urde /
\glb Mary is sure that it the-ACC Hans not annoy would /
\moregl
\gla seiner Freundin ihr$_i$ Herz auszusch\"utten. /
\glb his-DAT girlfriend-DAT her-ACC heart-ACC
{out to throw} /
\glc{‘Mary is sure that it would not annoy John to reveal her
heart to his girlfriend.’}
\endgl
\xe

Parameters can be set locally, as above, to hold only inside a
particular example, or they can be set globally to accord with
your general preferences.
```

The following illustrates how parameter settings can affect the display. Compare (38) with (37) on the previous page.

(38)

Mary$_i$	ist	sicher,	dass	es	den	Hans	nicht	stören	würde
Mary	is	sure	that	it	the-ACC	Hans	not	annoy	would

seiner	Freundin	ihr$_i$	Herz	auszuschütten.
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.’

Parameters can be set locally, as above, to hold only inside a particular example, or they can be set globally to accord with your general preferences.

---

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 `glspace`, which *ExPex* uses to set one of the parameters used in typesetting glosses. Executing the command

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

results in the definition (or redefinition) of the macro `\ling@glspace` so that it expands to the value `1.3em`. The macro `\begingl`, which is used to typeset glosses, uses `\ling@glspace`, typesetting the gloss so that the minimal interword separation is the dimension which `\ling@glspace` expands to. But *ExPex* users never have to concern themselves with the macro `\ling@glspace`. If they are not satisfied with the default spacing, they simply have to know that `glspace` is the key to setting the word spacing in glosses.

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

`\lingset{`key_1=$value_1$, . . . , key_n=$value_n$`}`

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 parser by putting the value in braces. The braces are removed by the parser.

Several *ExPex* macros take an optional argument, delimited by brackets, which is passed to `\lingset`. `\begingl`, used to typeset glosses, takes an optional argument. You might say, for example,

```
\begingl[glspace=.9em,everyglc=\it]
```

The optional argument of `\begingl` will be passed to `\lingset` and the result evaluated, so that the gloss 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, `everyglc=\it` will result in an italicized translation.

ExPex has various kinds of keys. The distinctions depend on the effect of executing (39) and the restrictions on the possible values which can appear.

(39) `\lingset{`*key*=*value*`}`

Command key: *value* is stored in the macro `\ling@`*key*. More precisely, after (39) is executed the macro `\ling@`*key* expands to *value*.

Incremental dimension parameter: *value* must be a dimension or a dimension prefixed by `!`. If *value* is a dimension, it is stored in `\ling@`*key*. If it is a !-prefixed dimension, the prefixed dimension is added to its former dimension and the result stored in `\ling@`*key*. If `\ling@`*key* was undefined or did not expand to a dimension, a fatal error results. Incremental parameters are very useful if minor adjustments to the format are desired.

Choice parameter: *value* must be drawn from a prescribed list. Although the *ExPex* choice parameters do store *value* when (39) is executed, the main purpose of executing (39) 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.

Pseudo parameter: (39) is executed for its side effects. *value* is not stored. The key `samplelabel` illustrates this. When `\lingset{samplelabel=A.}` is executed, for example, the macro `\ling@samplelabel` is not defined. Instead, the parameter `labelwidth` is set to the width of "A." in the current font.

In the interest of clarity, the exposition above was somewhat oversimplified (i.e. told some lies, but small lies). The macro in which a value associated with *key* is stored was assumed to be `\ling@`*key*. In fact, some keys use `\ling`*key*, with no `@` in the macro name so that they are easier for the user to access. In those cases in which it appeared that there is a significant chance that some users will want easy access to the value of a parameter at some point, the macro name `\ling`*key* was used rather than `\ling@`*key*.

When 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 one *ExPex* key (see Section 6.1, which will make clear why I am not giving its name) has a default value in this sense, but many *ExPex* keys are set in *expex.tex*. Since the distinction is not generally significant in what follows I will use the terms "default value" and "initial setting" interchangeably.

4. Simple examples

Macros:	`\ex~[], \xe, \lingnumoffset, \lingtextoffset`	
Parameters:	`numoffset=0pt`	(incrementable)
	`textoffset=1em`	(incrementable)
	`exskip`	(pseudo parameter)
	`aboveexskip=2.7ex plus .4ex minus .4ex`	(command)
	`belowexskip=2.7ex plus .4ex minus .4ex`	(command)
Counter:	`\excnt`	

The initial settings of parameters will be given in the section inventories, 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. If the value of a parameter is accessible by an external macro (i.e. of the form `\ling`*key*), the macro which expands to the value appears on the list of macros.

`\xe` terminates constructions initiated by `\ex`. The following sample paragraph illustrates the use of `\ex` . . . `\xe`. The convention in this guide 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.

(40) 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.

(41) 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.

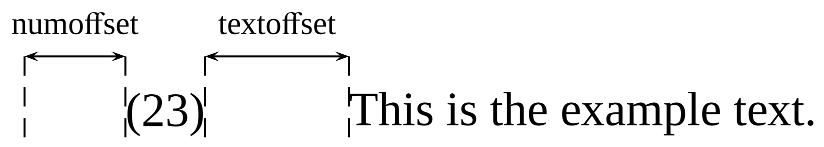
```
The following example is well-known from the literature on
parasitic gaps.  Here we are concerned with example formatting,
not with the interesting syntax.

\ex[numoffset=2em,textoffset=1ex,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, with `numoffset` measured from the left margin.



Example numbering is automatic. The count is kept in `\excnt`. It is incremented when `\ex` is evaluated, before the number is typeset. Vertical skip is inserted before and after examples, of amounts determined by the values of `aboveexskip` and `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`).

(42) Und hier können wir sehen was für Unfug wird gemacht wenn er einen ganz langen Satz binnen kriegt.

(43) 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 k\"onnen wir sehen was f\"ur 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 suppress a page break between the two examples. Since the vertical spacing between the examples is relevant to the discussion, it would not do to split the examples between pages. `\par\nobreak` is used to eliminate this possibility. `\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, so `\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. The pseudo parameter `specialexno`

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

(95) This is a crucial example.

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

(14) 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.

(96) * . . .

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

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

\ex[specialexno=14]
This is an example that was given many pages earlier.\xe

\noindent 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
```

Yhe regular sequence of example numbering is unaffected by an example whose number is assigned by `specialexno`.

`specialexno` does not have to be set to an integer.

(Δ) Earlier example.

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

(14, repeated) Earlier example.

```
\ex[specialexno={14, repeated}] Earlier example.\xe
```

Note the use of braces to hide the comma in the setting of `specialexno`. Otherwise, the key-value parser would get confused, interpreting 14 as the setting of `specialexno` and reporting that `repeated` is an undefined key.

Section (9.2) will show how to name example numbers, so that `specialexno` can be set by giving the name of the example number.

5. Examples with labeled parts

Macros:	`\pex~[], \nopreamble, \a, \linglabeloffset`	
Parameters:	`labeloffset=1em`	(incrementable)
	`labeltype=alpha`	(choice)
	`preambleoffset=1em`	(incrementable)
	`belowpreambleskip=1ex plus .2ex minus .2ex`	(command)
	`interpartskip=1ex plus .2ex minus .2ex`	(command)

Typical examples are given below, with the initial setting of the parameters.

(44) a. This is the first example.

b. This is the second example.

(45) Multipart examples often have titles.

a. This is the first example.

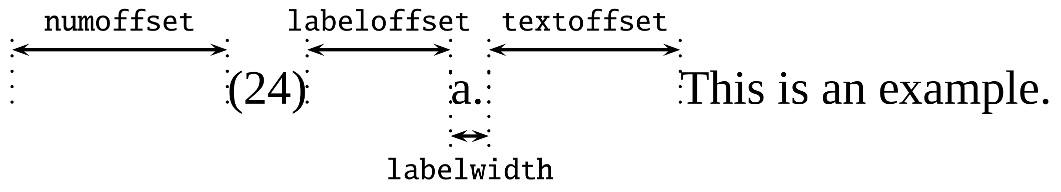
b. This is the second example.

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

\pex~ Multipart examples often have titles.
\a This is the first example.
\a This is the second example.
\xe
```

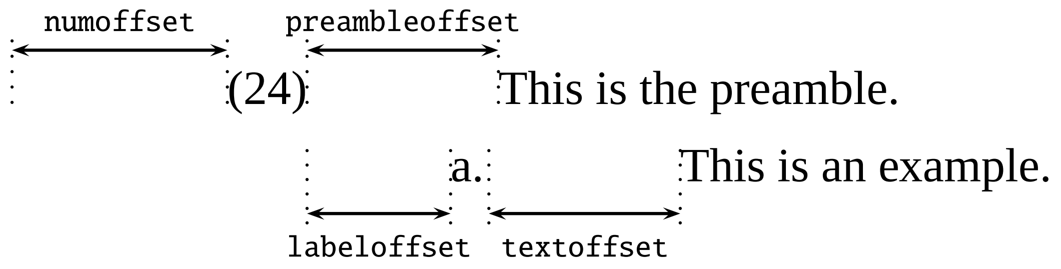
`\pex` must be closed by `\xe`, just like `\ex`. The macro `\a`, which introduces each labeled part, is defined only within `\pex` . . . `\xe`. Visible material which occurs before the first labeled entry, as in (44), is called the preamble. Although the initial settings produce the format in (44), the offset of the preamble can be set independently of the offset of the labels.

Visible material which occurs before the first labeled entry, as in (45), is called the preamble. The parameters governing multipart example displays which do not have a preamble are shown below. `numoffset` is measured from the left margin. All of the parameters that apply in `\ex` constructions also apply in `\pex` constructions.

(46)


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 should a good approximation to the the width of “a.” and “1.” in the current font, unless your font is unusual. If the labels are alphabetic, wide characters like “w” will spill over the label noticably and the label width might need adjustment, depending on the width of the slot it spills into. If the labels are numeric and and numbers greater than 9 are needed, the labels will spill over substantially and `labelwidth` will almost certainly need adjustment if high quality output is desired. We will return to this issue later.

Although the initial settings produce the format in (44), the offset of the preamble can be set independently of the offset of the labels. The layout of multipart displays with a preamble is shown below. (`labelwidth` is omitted to reduce clutter.)

(47)


Contrast (48) and (49). In (49), `labeloffset` and `preambleoffset` have different settings.

(48) *Principles of the Theory of Binding*

A. An anaphor is bound in its governing category.

B. A pronomial is free in its governing category.

C. An R-expression is free.

(49) *Principles of the Theory of Binding*

A. An anaphor is bound in its governing category.

B. A pronomial is free in its governing category.

C. An R-expression is free

```
\pex[labeltype=caps]
{\it Principles of the Theory of Binding}
\a An anaphor is bound in its governing category.
```

```
\a A pronomial is free in its governing category.
\a An R-expression is free.
\xe

\pex~[labeltype=caps,labeloffset=!.8em]
{\it Principles of the Theory of Binding}
\a An anaphor is bound in its governing category.
\a A pronomial is free in its governing category.
\a An R-expression is free
\xe
```

`labeloffset` is incrementable. Assuming that `preambleoffset` and `labeloffset` initially have their default settings, which are the same, the effect of `labeloffset=!.8em` is to offset the labels .8 em more than the preamble.

The setting of `belowpreambleskip` determines the extra vertical skip between the preamble and the first labeled entry; `interpartskip` determines the extra vertical skip between labeled entries.

(50) *Principles of the Theory of Binding*
A. An anaphor is bound in its governing category.
B. A pronomial is free in its governing category.
C. An R-expression is free.

```
\pex[labeltype=caps,belowpreambleskip=0pt,interpartskip=0pt]
{\it Principles of the Theory of Binding}
\a An anaphor is bound in its governing category.
\a A pronomial is free in its governing category.
\a An R-expression is free.
\xe
```

5.1. Refinements

Macro:	`\nopreamble`	
Counter:	`\pexcnt`	
Parameter:	`everylabel`	(command)

Suppose you want to produce a display like the following, perhaps because statements a–c have already been discussed.

(51) d. Fourth statement.
e. Fifth statement.
f. Sixth statement.

Automatic labeling uses the counter `\pexcnt`. It is incremented just before the label is typeset. So you might try the code:

```
\pex[interpartskip=0pt]
\advance\pexcnt by 3
\a Fourth statement.
\a Fifth statement.
\a Sixth statement.
\xe
```

This code runs into a problem. `\pex` interprets the second line as a preamble. `\pex` is not clever enough to figure out that this material is nonprinting. Consequently, a blank preamble will be typeset. A diacritic is needed to tell `\pex` that there is no preamble. The rule is this. After `\pex` checks for a following optional tilde and processes the optional argument (if there is one), it looks at the next nonspace item. If it is `\a` or `\nopreamble`, it concludes there is no preamble. This is the only use for `\nopreamble`; a diacritic to tell `\pex` that pre-`\a` material should not be treated as a preamble. The code for (50) can therefore be written:

```
\pex[interpartskip=0pt]
\nopreamble \advance\pexcnt by 3
\a Fourth statement.
\a Fifth statement.
\a Sixth statement.
\xe
```

`everylabel` is provided in case part labeling like that in (52) or (53) is needed.

(52) *a.* An example
b. An example
c. An example

(53) 1. An example
2. An example
3. An example

```
\pex[everylabel=\it]
\a An example
\a An example
\a An example
\xe

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

`samplelabel` will be discussed shortly.

`\ling@everylabel` is inserted before the character or integer generated from `\pexcnt` is typeset; grouped, so that the effects of commands like `\it` are localized to the label.

5.2. Adjusting the label width and alignment

Parameters:	`labelwidth=.78em`	(incrementable)
	`labelalign=left`	(choice)
	`samplelabel`	(pseudo parameter)

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`. The part labels can be uppercase letters, lowercase letters, or integers. The choice is made via by the parameter `labeltype`, which can be set to `caps`, `alpha`, or `numeric`. The setting of `interpartskip` determines the vertical skip which is inserted between the labeled entries.

For `\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 significant effect on the appearance. Individual tastes (and publisher's demands) may differ, but I prefer center alignment in these cases.

(54) *left aligned labels*
i. A typical example.
j. A typical example.
k. A typical example.
l. A typical example.
m. A typical example.
n. A typical example.

(55) *center aligned labels*
i. A typical example.
j. A typical example.
k. A typical example.
l. A typical example.
m. A typical example.
n. A typical example.

(56) *right aligned labels*
i. A typical example.
j. A typical example.
k. A typical example.
l. A typical example.
m. 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.

(57) *left aligned labels*
7. A typical example.
8. A typical example.
9. A typical example.
10. A typical example.
11. A typical example.
12. A typical example.

(58) *center aligned labels*
7. A typical example.
8. A typical example.
9. A typical example.
10. A typical example.
11. A typical example.
12. A typical example.

(59) *right aligned labels*
7. A typical example.
8. A typical example.
9. A typical example.
10. A typical example.
11. A typical example.
12. A typical example.

The default 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.

Particularly wide labels create another problem which needs attention. Recall that the slot in which labels are typeset does not automatically adjust its width to fit the widest label. It is fixed in advance by the setting of `labelwidth`. Wide labels will spill out of the label slot; to the right, left, or on both sides, depending on the setting of `labelalign`. Particularly in the case of double digit numeric labels, the width of the label slot needs to be changed from the default value. One can do this by saying something like `labelwidth=!1ex`, which will increase the label width by 1 ex. This gives more or less satisfactory results. The pseudo parameter `samplelabel` is provided to help in adjusting the label width. If you say, for example,

```
\lingset{samplelabel=10.}
```

then `labelwidth` is set to the width of “10.” in the current font. Examples (54–56) used `\lingset{samplelabel=m.}` and (57–59) used `\lingset{samplelabel=10.}`

It is a side issue, but the reader may have wondered how (54–56) and (57–59) were typeset. The idea is simple. You say:

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

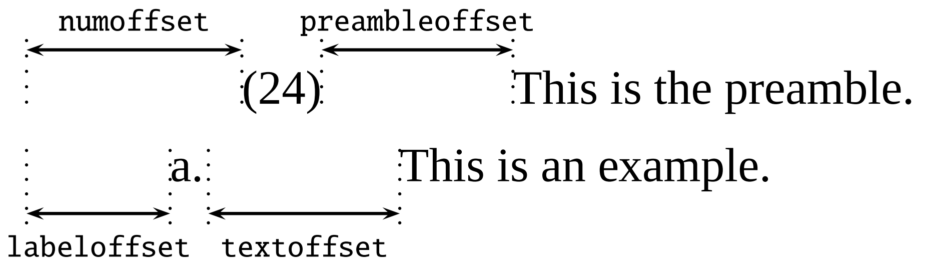
`\hss` is used to give a little stretch or shrink so that dimensional rounding does not lead to an under or overfull `\line{ ... }`.

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}}
```

5.3. The choice key `labelanchor`

Ordinarily, the label offset is measured from the right edge of the example number. There is little choice if a label occurs on the same line as the example number (i.e. there is no preamble). In example displays with a preamble, however, it is sometimes desirable to measure the labeloffset from the left margin, as shown below.

(60)


The most common application of anchoring the label offset at the left margin is in examples like the following. If it is impossible or undesirable to break a line (or graphic display of some kind), (62) is an alternative to (61).

(61) a. This is a very long example that will not fit in one line and cannot be broken for some reason.
b. A short line might be more interesting linguistically, but less interesting typographically.

(62)
a. This is a very long example that will not fit in one line and cannot be broken for some reason.
b. A short line might be more interesting linguistically, but less interesting typographically.

```
\pex
\a This is a very long example that will not fit in one line and
cannot be broken for some reason.
\a A short line might be more interesting linguistically, but less
interesting typographically.
\xe

\pex~[labelanchor=margin,labeloffset=0pt]\par
\a This is a very long example that will not fit in one line and
cannot be broken for some reason.
\a A short line might be more interesting linguistically, but less
interesting typographically.
\xe
```

The choice key `labelanchor` can be set to `margin`, `numleft`, or `numright`. `numright` is the default; which most users will never reset. Note that the preamble in the second `\pex` is `\par`. `\relax` would accomplish the same thing. Something must come before the first `\a`, otherwise `\pex` will think there is no preamble.

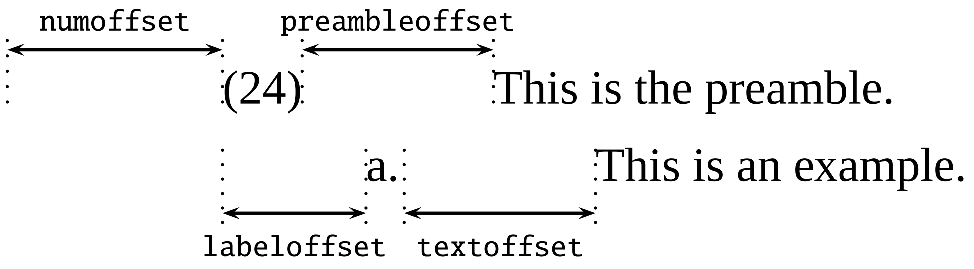
Another way to handle situations like this is to use `\ex` and insert explicit labels. Something like the following works well.

```
\ex \smallskip \leftskip=0pt
a.\enspace This is a very long example that will not fit in one
line and cannot be broken for some reason.
\smallskip
b.\enspace A short line might be more interesting linguistically,
but less interesting typographically.
\xe
```

(63)
a. This is a very long example that will not fit in one line and cannot be broken for some reason.
b. A short line might be more interesting linguistically, but less interesting typographically.

If you can think up a title, so that the top line contains more than the example number, the appearance is improved.

Anchoring the label offset at the left edge of the example number (setting `labelanchor` to `numleft`) is unlikely to be particularly useful. The possibility is included mostly for the sake of completeness. The setup is shown below.

(64)


6. Further control over the format inside `\ex` and `\pex` constructions

(This section can be omitted at first reading.)

Apart from the proper indentation of example numbers, part numbers, and example text itself, there are several reasons why you might want typesetting within examples to be governed by somewhat different rules than typesetting outside examples. The most important is probably page breaking, which is generally much more disruptive if it occurs inside an example than if it occurs in a paragraph of text. Steps should be taken to minimize the possibility that examples are disrupted by page breaks.

6.1. Preventing page breaks in examples

Parameters: `exnosplitfil=0pt plus .3\vsize`
`exnosplitpenalty=300`

Tex carries out page breaking by assigning a cost (via penalties) to each possible page break and choosing the lowest cost option. The first two parameter settings above inform Tex that when it encounters `\ex` or `\pex`, at a cost of 300, it can file up the page with white space that is 30% the vertical size of the page and start the example on the next page. The cost is substantial, but will be paid if it is the best option. If the example is sizable enough, *Tex* will choose this option, avoiding an example that is split between pages, at the cost of some white space at the bottom of the page which precedes the example. The default settings of `exnosplitfil` and `exnosplitpenalty` are chosen to balance the desirablities of keeping examples intact and minimizing the occurrence of pages which appear to be undersize. (Undersize from the readers perspective. *Tex* doen't think the page is undersize because vertical fill has been inserted to fill out the page.)

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

Macro:	`\goodpar`	
Parameters:	`goodparfil=0pt plus .15\vsize`	(command)
	`goodparpenalty=-100`	(command)

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 (65).

(65) a. First example.
b. Second example.
c. Third example.
d. Fourth example.
e. Fifth example.
f. Sixth example.
g. Seventh example.
h. Eighth example.

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 First example.\par\nobreak
\a Second example.
\a Third example.\par\nobreak
\a Fourth example.\par\nobreak
\a Fifth example.\par\nobreak
\a Sixth example.
\a Seventh example.\par\nobreak
\a Eighth example.
\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.

ExPex provides the macro `\goodpar` which can help in such situations. It is essentially a paragraph break which provides a certain amount of vertical fill (determined by the setting of `goodparfil`) if a break is made at that point. Page breaking is encouraged by assigning a reward (a negative penalty, determined by the setting of `goodparpenalty`) to a page break triggered by `\goodpar`. The code below will generally be satisfactory if appropriate values (the default values, for example) are given to the goodpar parameters.

```
\pex[interpartskip=.25ex]
\a First example.
\a Second example.\goodpar
\a Third example.
\a Fourth example.
\a Fifth example.
\a Sixth example.\goodpar
\a Seventh example.
\a Eighth example.
\xe
```

Although `\goodpar` was intended to help in formating examples, it can be used anywhere in your document.

The default values of `exnosplitfil` and `goodparfil` are fairly large, more suited to drafts and informal papers than camera-ready copy. But it is easy to change the global values if you need to go to greater lengths to prevent whitespace from appearing at the bottom of pages.

6.3. The parameters `everyex` and `Everyex`

Aside from 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 inside examples.

The parameters `everyex` and `Everyex` are provided by *ExPex* to facilitate making these kind of adjustments inside examples. The effect of

`\lingset{everyex=`*value*`}`

is to define a macro `\ling@everyex` which expands to *value*. `Everyex` works in a similar fashion. When `\ex` is expanded, a group is first initiated, then `\ling@Everyex` is expanded, then the parameter settings furnished to `\ex` are passed to `\lingset`, then `\ling@everyex` is executed.

In order to illustrate, this consider the code below:

```
\ex[textoffset=3em,
   everyex={\hsize=3in \it \advance\baselineskip by 2pt}]
Und hier k\"onnen wir sehen was f\"ur Unfug wird gemacht
wenn er einen ganz langen Satz binnen kriegt.
\xe
```

The XKV evaluator will remove surrounding braces from values associated with keys. The braces around the value associated with `everyex` above are not required by the syntax, but do make the code much more readable. If a value contains a comma, however, braces are mandatory. The XKV evaluator uses commas to parse the key/value pairs and a comma inside a value which is not hidden by braces will be incorrectly interpreted as separating key/value pairs. Chaos will ensue.

The code above produces:

(66) *Und hier können wir sehen was*
für Unfug wird gemacht wenn er
einen ganz langen Satz binnen
kriegt.

Since the following code produces the same thing, the utility of the parameter `everyex` is not yet evident. It does have the advantage that all the special settings are put in one place, the optional argument of `\ex`. More substantial advantages will become clearer after the parameter `lingstyle` is discussed in the next section.

```
\ex[textoffset=3em]
\hsize=3in \it \advance\baselineskip by 2pt
```

```
Und hier k\"onnen wir sehen was f\"ur Unfug wird gemacht
wenn er einen ganz langen Satz binnen kriegt.
\xe
```

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 6.5 in and that your final aim is to produce camera-ready copy with an hsize of 4.375 in. Suppose also that you are at editing stage where you want to see exactly what the examples will look like in the finished product, but still want full width double spaced text. One way to accomplish this is to say

```
\lingset{everyex={\hsize=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.

The parameter `Expex` is provided to accomodate this situation. The intention is that special settings that should hold throughout the document are assigned to `Expex`, with `expex` reserved for local variations.

6.4. The parameter `lingstyle`

Macro:	`\definelingstyle`	
Parameter:	`lingstyle`	(pseudo parameter)

After

`\definelingstyle{`*name*`}{`key_1`=`$value_1$`,` … `,`key_n`=`$value_n$`}`

then

`\lingset{lingstyle=`*name*`}`

is equivalent to

`\lingset{`key_1`=`$value_1$`,` … `,`key_n`=`$value_n$`}`

If you have multiple special examples in your document which all require the same special parameter settings, the array of parameter settings (perhaps only a single change) which is required can be given a style name and the parameter changes can be simply made.

Suppose, for example, that you have multiple examples that you want to format in the same way as (66). If you first define the style “narrow italic” by:

```
\definelingstyle{narrowitalic}{textoffset=3em,
   everyex={\hsize=3in \it \advance\baselineskip by 2pt}}
```

Then examples like (66) can be simply formated by using:

```
\ex[lingstyle=narrowitalic]
```

Aside from relatively trivial benefit of saving typing, the ability to define styles has all the well-known advantages of uniformity, ease of modification, and clarity of typographic organization. `lingstyle` is a parameter like all others in that it can be set along with other parameters with standard left to right key setting.

You could say, for example:

```
\ex[lingstyle=narrowitalic,textoffset=1em]
Und hier k\"onnen wir sehen was f\"ur Unfug wird gemacht
wenn er einen ganz langen Satz binnen kriegt.
\xe
```

(67) *Und hier können wir sehen was für*
Unfug wird gemacht wenn er einen
ganz langen Satz binnen kriegt.

7. Judgment marks

Macros: `\judge, \ljudge`
Parameters: `*` (pseudo parameter)

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

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

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

In my view (69), with a little whitespace inserted between the asterisk and the example sentence, looks somewhat better than (68).

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

(69) * 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`...

Multi-part examples are more complex, if alignment is to be maintained. (70) is not satisfactory.

(70) 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

(71) 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.

```
\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
```

(72) 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
```

(73) 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 (72) is too large, `textoffset` can be adjusted directly, so you could write

```
\pex[textoffset=!.7em]
\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
```

(74) 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.

8. Glosses

Macros:	`\begingl[]`, `\endgl`, `\gla`, `\glb`, `\glc˜`, `\moregl`	
Parameters:	`glspace=.6em`	(command)
	`everygla=\it`	(command)
	`everyglb=`	(command)
	`everyglc=`	(command)
	`aboveglcskip=.5ex`	(command)
	`moregloffset=0pt`	(command)
	`abovemoreglskip=.25ex`	(command)

The keys `everyglb` and `everyglc` are initially set to empty. This is accomplished by

```
\lingset{everyglb=,everyglc=}
```

We start with a straightforward example.

(75) *Il*	*semble*	*au*	*général*	*être*	*arrivé*	*deux*	*soldats*	*en*	*ville.*
there	seems	to the	general	to be	arrived	two	soldiers	in	town

‘There seem to the general to have arrived two soldiers in town.’

```
\ex
\begingl
\gla Il semble au g\'en\'eral \^etre arriv\'e deux soldats en ville. /
\glb there seems {to the} general {to be} arrived two soldiers in
```

```
town /
\glc{‘There seem to the general to have arrived two soldiers in
town.’}
\endgl
\xe
```

`\begingl` . . . `endgl` constructs a table, using `\halign`. `glspace` is the column separation. `\gla` and `\glb` parse the line up to the slash as a sequence of space delimited items. Note that a space must precede the slash to terminate the parse and that multiple word glosses like “to the” and “to be” must be grouped to hide the internal space from the parsing process. `\glc` takes an argument which it puts into a single cell, hiding its width. Extra vertical skip determined by `aboveglcskip` is inserted above the `\glc` line. `\ling@everygla` is inserted in the `\gla` line cells, `ling@everyglb` in the `\glb` line cells, and `ling@everyglc` in the single `\glc` line cell.

If the code above is preceded by:

```
\lingset{glspace=1.2em,everygla=\rm,everyglb=\sl,everyglc=\it,
   aboveglcskip=1ex}
```

then we get

(76)	Il	semble	au	général	être	arrivé	deux	soldats	en	ville.
	there	*seems*	*to the*	*general*	*to be*	*arrived*	*two*	*soldiers*	*in*	*town*

‘There seem to the general to have arrived two soldiers in town.’

The next example illustrates the use of the macro `\moregl` and the parameters `moregloffset` and `abovemoreglskip`.

(77) a.

Mary$_i$	*ist*	*sicher,*	*dass*	*es*	*den*	*Hans*	*nicht*	*stören*	*würde*
Mary	is	sure	that	it	the-ACC	Hans	not	annoy	would

seiner	*Freundin*	*ihr$_i$*	*Herz*	*auszuschütten.*
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*
Mary	is	sure	that	his-DAT	girlfriend-DAT	her-ACC	heart-ACC

auszuchütten	*dem*	*Hans*	*nicht*	*schaden*	*würde.*
out to throw	the-DAT	Hans	not	damage	would

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

```
\pex[moregloffset=1em,abovemoreglskip=1ex]
\nopreamble
\def\\#1{-{\tenrm #1}}%
\a
\begingl
\gla Mary$_i$ ist sicher, dass es den Hans nicht st\"oren w\"urde /
```

```
\glb Mary is sure that it the\\{ACC} Hans not annoy would /
\moregl
\gla seiner Freundin ihr$_i$ Herz auszusch\"utten. /
\glb his\\{DAT} girlfriend\\{DAT} her\\{ACC} heart\\{ACC}
{out to throw} /
\glc {‘Mary is sure that it would not annoy John to reveal her
heart to his girlfriend.’}\endgl
\a
\begingl
\gla Mary$_i$ ist sicher, dass seiner Freunden ihr$_i$ Herz /
\glb Mary is sure that his\\{DAT} girlfriend\\{DAT} her\\{ACC}
heart\\{ACC} /
\moregl
\gla auszuch\"utten dem Hans nicht schaden w\"urde. /
\glb {out to throw} the\\{DAT} Hans not damage would /
\glc{‘Mary is sure that to reveal her heart to his girlfriend
would not damage John.’}
\endgl
\xe
```

The example is from an article by Idan Landau (NLLT, 19.1, 2001).

If you need more than one line for the translation in a gloss, using `\glc` for both lines of the translation will insert an unfortunate vertical skip above the second line. Following the convention established for `\ex` and `\pex`, no skip is inserted if `\glc` is followed by tilde. So use `\glc` for the first line and `\glc~` for the second.

9. Tables in examples

Macro: `\tspace[], \exnoprint, \crnb`

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`.

Many tabular examples have the form

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

For example:

(78)	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&& \qquad #\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 (78), with a title and labeled columns, is given below. `\hwit` is described below.

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

root	*perfect stem*	*gloss*
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
```

`\hwit` (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.

9.1. Tables with labeled lines

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

If reference must be made to particular lines in (79), the lines need labels of some sort. One approach is to explicitly enter the line labels a–e:

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

	root	*perfect stem*	*gloss*
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\hwit{root}& \hfil\hwit{perfect stem}&
   \hfil\hwit{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 (80) 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 changed to match the new labeling. We will see in Section 9.2 that lines in tables can be named and reference made by name, using implicit line numbering.

9.2. Some useful table making tools

Macros:	`\lingdima, \lingdimb, \lingdimc, \tspace[], \crs`	
Parameters:	`dima, dimb, dimc`	(command, no default)
	`crskip=.6em`	(command)

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\ling`*tag*, so that horizontal skip like `\tspace[dima]` or `\tspace[textoffset]` can be easily used in table construction. The code for (80) 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. `\ling@crskip` expands to the setting of `crskip`, and

`\crs` $\rightarrow$ `\cr \noalign{\vskip\ling@crskip}`

9.3. Tables that can break between pages

Macro: `\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, `\ling@everyex` and `\ling@Everyex` 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`or `\exdisplay[ ... ]\noexno` to cancel `\excnt` advancement.

Now consider the 'tall display' (81). 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.

(81) High vowel in the nucleus of the root

	root	*perfect stem*	*gloss*
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}` at a few points to prevent a page break between a heading and the remainder of the table which follows. Page breaks are encouraged by `\goodpar` at a few points where it is judged that the logic of the table can tolerate it. 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                    % gloss
(\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{\goodpar}
&& suap& \\su-suap& sleep\cr
&& miaks& \\mi-miaks& glitter\cr
&& auc& \\u-auc& please\cr
\noalign{\goodpar\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{\goodpar}
&& skand& \\ka-skand& leap\cr
&& mard& \\ma-mard& rub, crush\cr
&& mnaa& \\ma-mnaa& note\cr
}\xe
```

9.4. 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'.

(82) *Present Indicative*

	active			*middle*		
	sg	*du*	*pl*	*sg*	*du*	*pl*
1	**dvéṣ**-mi	dviṣ-vás	dviṣ-más	dviṣ-é	dviṣ-váhe	dviṣ-máhe
2	**dvék**-ṣi	dviṣ-ṭhás	dviṣ-ṭhá	dvikṣ-é	dviṣ-ā́the	dviḍ-ḍhvé
3	**dvéṣ**-ṭi	dviṣ-ṭás	dviṣ-ánti	dviṣ-ṭé	dviṣ-ā́te	dviṣ-áte

Assuming that the page width has been set to 4.3 in (`\hsize=4.3in`), the following code produces (82), which is precisely 4.3 in wide.

```
\exdisplay[labeloffset=.5em,dima=.6em,textoffset=.6em]
\def\\#1{$\acute{\hbox{\=#1}}$}%
\tabskip=0pt \openup1pt
\halign to \hsize{\tspace[labeloffset]#\tspace[textoffset]\hfil&
   # \hfil\tabskip=0pt plus 1fil&
   # \hfil& # \hfil& \tspace # \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
```

10. Referring to examples and labeled parts of examples

10.1. Unnamed reference

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

These produce the last, next, before last, after next, and before before last example numbers. The example number is set early in the expansion of `\ex` and `\pex`, so these macros can be used inside an example, with "last example" taking on the meaning of "current example" and the others similarly interpreted.

It is 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 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. If an intervening example is deleted or added, no problem arises. If the example which is referred to is deleted, then *Tex* can report a missing reference.

10.2. Named reference

Parameters: `tag` (pseudo parameter)
Macros: `\deftag, \deftagex, \deftaglabel, \deftagpage, \a<>`
`\getref, \getfullref, \refproofing`

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 `label-type`, 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 (83), with the deftag commands invisible.

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

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

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

`\getref{snoopy}`	→	83
`\getref{snoopy.dog}`	→	b
`\getfullref{snoopy.dog}`	→	83b
`\getref{snoopy.a}`	→	c
`\getfullref{snoopy.a}`	→	83c
`\getref{dog}`	→	undefined tag warning, or spurious reference

A spurious reference is produced 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.

If the label type in (83) is changed to `numeric`, `\getfullref{snoopy.dog}` will expand to 83.2, with a period separating the example number and the part label. The ability of `\getfullref` to make the correct decision about whether or not to separate the example number from the part label by a period is limited. If `everylabel` is nonempty, a period is inserted unless the value of `everylabel` happens to begin with the control sequence `\char`, which is unlikely. In reference to the first part of (53), for example, `\getfullref{`*tag*$_1$`.`*tag*$_2$`}` produces 53.A1, assuming that `\deftagex{`*tag*$_1$`}` and `\deftaglabel{`*tag*$_2$`}` have been used, either explicitly or implicitly, to tag the example and part. If you want 53A1, you can use `\getref{`*tag*$_1$`}\getref{`*tag*$_1$`.`*tag*$_2$`}`.

There are two shortcuts to writing the code for (83) which, unlike most shortcuts, make the code easier to read as well. One can write:

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

<. . . > delimits an optional argument which is interpreted as either a tag of the example number, or of the label, depending on where it occurs.

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.

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 over lined. Highlighting is a significant aid in copy editing. The difference is illustrated below.

(84)	a.	Consider (83b), for example.	default behavior
	b.	Consider $\boxed{\text{(83b)}}$, for example.	`\refproofing`, *PSTricks* available
	c.	Consider ($\overline{\underline{\text{83b}}}$), for example.	`\refproofing`, *PSTricks* not available

10.3. 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 `\gatherftags`, 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.

10.4. 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.

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

\noindent You can then refer to (\getref{snoopy}) later in your
document.
```

(85) a. First Example.

b. Second Example.

c. Third Example.

You can then refer to (85b) later in your document.

`\lastlabel` expands to the most recent label, in the form in which it is printed (i.e. containing the expansion of `\everylabel`).

You can also say:

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

(85b) Second example.

An alternative is:

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

(85b, repeated) Second example.

Note that 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
```

(86) 1. First Example.

2. Second Example.

3. Third Example.

then, if you want to repeat (86.2) at some point, you can use

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

(86.2) Second example

When `\refproofing` is in force; if `\getref`, `\getfullref`, or `\lastx` are used inside `\deftag` or as values which set `specialexno`, *reference highlighting is turned off* and gentle warnings about undefined tags is not in effect. In these contexts, minimal versions of `\getref`, `\getfullref`, and `\lastx` are used that strip away testing for tag definition and highlighting. Otherwise, their expansions would be much too complex to be usuable in these contexts. Since no testing is done to first check if a tag is defined, an undefined tag will not produce a user friendly warning message, simply a *Tex* crash of the familiar kind. If the tag `foo` is undefined, for example, and you try to evaluate its reference, *Tex* will crash and inform you that `\lingtag@foo` is undefined.

10.5. 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:

```
1 \def\currsec{\the\chapterno
2    \ifnum\secno>0 .\the\secno
3    \ifnum\subsecno>0 .\the\subsecno
4    \ifnum\subsubsecno>0 .\the\subsubsecno \fi\fi\fi}
5 \def\deftagsec#1{\deftag\currsec{#1}}
```

This assumes that there are counters `\chapterno`, `\secno`, `\subsecno`, and `\subsecsecno` and that when a chapter is initiated, `\secno` is set to 0, when a section is initiated, `\secsecno` 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.)

11. *ExPex* and *PSTricks*

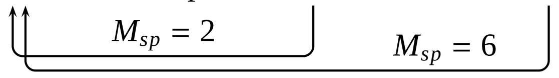
Several features of *ExPex*, listed in (87), come into play only if *pstricks.tex* has been loaded at the point that *expex.tex* is loaded. They are intended to make it easier to use *PSTricks* in examples.

(87)
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 (88).

(88) 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 *PSTtricks* parameters `angle`, `arrows`, `nodesep`, `labelsep`, and `linearc`, and the *ExPex* parameter `dima`. Note also that the *ExPex* scratch dimension `\lingdima` is used seemlessly by the *PST-Node* macros.

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

```
\ex
\psset{angle=-90,nodesep=0pt,arrows=->,dima=.2em,labelsep=.25ex,
   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.

Index of control sequences, parameters, and special symbols