Recording and cross-referencing document properties*

The LaTeX Project†

January 25, 2026

Abstract

This code implements command to record and (expandably) reference document properties. It extends the standard `\label`/`\ref`/`\pageref` commands.

Contents

*This module has version v1.0l dated 2026-01-16, © The LaTeX Project.

†E-mail: latex-team@latex-project.org

1 Introduction

The module allows to record the "current state" of various document properties (typically the content of macros and values of counters) and to access them in other places through a label. The list of properties that can be recorded and retrieved are not fix and can be extended by the user. The values of the properties are recorded in the `.aux` file and can be retrieved at the second compilation.

The module uses the ideas of properties and labels. A label is a document reference point: a name for the user. An property is something that LaTeX can track, such as a page number, section number or name. The names of labels and properties may be arbitrary. Note that there is a single namespace for each.

2 Design discussion

The design here largely follows ideas from zref. In particular, there are two independent concepts: properties that can be recorded between runs, and labels which consist of lists of these properties. The reason for the split is that individual labels will want to record some but not all properties. For examples, a label concerned with position would track the x and y coordinates of the current point, but not for example the page number.

In the current implementation, properties share a single namespace. This allows multiple lists to re-use the same properties, for example page number, absolute page number, etc. This does mean that *changing* a standard property is an issue. However, some properties have complex definitions (again, see zref at present): having them in a single shared space avoids the need to copy code.

Labels could be implemented as `prop` data. That is not done at present as there is no obvious need to map to or copy the data. As such, faster performance is available using a hash table approach as in a "classical" set up. Data written to the `.aux` file uses simple paired *balanced text* not keyvals: this avoids any restrictions on names and again offers increased performance.

The expl3 versions of the label command do not use `\@bsphack`/`\@esphack` to avoid double spaces, but the LaTeX 2_ε command does as it lives at the document command level.

The reference commands are expandable.

Currently the code has nearly no impact on the main `\label` and `\ref` commands as too many external packages rely on the concrete implementation. There is one exception: the label names share the same namespace. That means that if both `\label{ABC}` and `\RecordProperties{ABC}{page}` are used there is a warning `Label 'ABC' multiply defined`.

3 Handling unknown labels and properties

With the standard `\label`/`\ref` commands the requested label is either in the `.aux`-file (and so known) or not. In the first case the stored value can be used, in the second case the reference commands print two question marks.

With flexible property lists a reference commands asks for the value of a specific property stored under a label name and we have to consider more variants:

- If the requested property is unknown (not declared) the system is not correctly set up and an error is issued.

- If the label is unknown, the default of the property is used.
- If the label is known, but doesn't provide a value for the property then again the default of the property is used.
- The command `\property_ref:nnn` allows to give a local default which is used instead of the property default in the two cases before.

4 Rerun messages

As the reference commands are expandable they can neither issue a message that the label or the label-property combination is unknown, nor can they trigger the rerun message at the end of the LaTeX run.

Where needed such messages must therefore be triggered manually. For this two commands are provided: `\property_ref_undefined_warn:` and `\property_ref_undefined_warn:nn`. See below for a description.

5 Open points

- The `xpos` and `ypos` properties require that the position is stored first but there is no (public) engine independent interface yet. Code must use `\tex_savepos:D`.

6 Code interfaces

`\property_new:nnnn` `\property_gset:nnnn`

`\property_new:nnnn {⟨property⟩} {⟨setpoint⟩} {⟨default⟩} {⟨code⟩}`
`\property_gset:nnnn {⟨property⟩} {⟨setpoint⟩} {⟨default⟩} {⟨code⟩}`

LaTeX 2_ε-interface: see `\NewProperty`, `\SetProperty`.
Sets the ⟨*property*⟩ to have the ⟨*default*⟩ specified, and at the ⟨*setpoint*⟩ (either `now` or `shipout`) to write the result of the ⟨*code*⟩ as part of a label. The ⟨*code*⟩ should be expandable. The expansion of ⟨*code*⟩ (the value of the property) is written to the `.aux` file and read back from there at the next compilation. Values should assume that the standard LaTeX catcode régime with `@` a letter is active then.

If the property is declared within a package it is suggested that its name is build from letters, hyphens and slashes, and is always structured as follows:
⟨*package-name*⟩`/`⟨*property-name*⟩.

`\property_record:nN` `\property_record:nn` `\property_record:(nV|ee)`

`\property_record:nN {⟨label⟩} ⟨clist var⟩`
`\property_record:nn {⟨label⟩} {⟨clist⟩}`

LaTeX 2_ε-interface: see `\RecordProperties`.
Writes the list of properties given by the ⟨*clist*⟩ to the `.aux` file with the ⟨*label*⟩ specified.

`\property_ref:nn ⋆` `\property_ref:ee ⋆`

`\property_ref:nn {⟨label⟩} {⟨property⟩}`

LaTeX 2_ε-interface: see `\RefProperty`.
Expands to the value of the ⟨*property*⟩ for the ⟨*label*⟩, if available, and the default value of the property otherwise. If ⟨*property*⟩ has not been declared with `\property_new:nnnn` an error is issued. The command raises an internal, expandable, local flag if the reference can not be resolved.

\property_item:nn ⋆ \property_item:nn {⟨label⟩} {⟨property⟩}
\property_item:ee ⋆

New: 2025-11-20

Retrieves the value of the ⟨property⟩ for the ⟨label⟩ like \property_ref:nn but the result is returned within the \unexpanded primitive (\exp_not:n), which means that the ⟨value⟩ does not expand further when appearing in an e-type or x-type argument expansion. This allows for example to handle values containing user commands which are not safe in an expansion context.

\property_ref:nnn ⋆ \property_ref:nnn {⟨label⟩} {⟨property⟩} {⟨local default⟩}
\property_ref:een ⋆

LaTeX 2ε-interface: see \RefProperty.
Expands to the value of the ⟨property⟩ for the ⟨label⟩, if available, and to ⟨local default⟩ otherwise. If ⟨property⟩ has not been declared with \property_new:nnnn an error is issued. The command raises an internal, expandable local flag if the reference can not be resolved.

\property_ref_undefined_warn: \property_ref_undefined_warn:

LaTeX 2ε-interface: not provided.
Triggers the standard warning

```
LaTeX Warning: There were undefined references.
```

at the end of the document if there was a recent \property_ref:nn or \property_ref:nnn which couldn't be resolved and so raised the flag. "Recent" means in the same group or in some outer group!

\property_ref_undefined_warn:n \property_ref_undefined_warn:n {⟨label⟩}
\property_ref_undefined_warn:e

LaTeX 2ε-interface: not provided.
Triggers the standard warning

```
LaTeX Warning: There were undefined references.
```

at the end of the document if ⟨label⟩ is not known. At the point where it is called it also issues the warning

```
Reference '⟨label⟩' on page ⟨page⟩ undefined.
```

\property_ref_undefined_warn:nn \property_ref_undefined_warn:nn {⟨label⟩} {⟨property⟩}
\property_ref_undefined_warn:ee

LaTeX 2ε-interface: see \RefUndefinedWarn.
Triggers the standard warning

```
LaTeX Warning: There were undefined references.
```

at the end of the document if the reference can not be resolved. At the point where it is called it also issues the warning

```
Reference '⟨label⟩' on page ⟨page⟩ undefined
```

if the label is unknown, or the more specific

```
Property '⟨property⟩' undefined for reference '⟨label⟩' on page ⟨page⟩
```

if the label is known but doesn't provide a value for the requested property.

\property_if_exist_p:n ⋆ \property_if_exist_p:n {⟨property⟩}
\property_if_exist_p:e ⋆ \property_if_exist:nTF {⟨property⟩} {⟨true code⟩} {⟨false code⟩}
\property_if_exist:n*TF* ⋆
\property_if_exist:e*TF* ⋆

LaTeX 2ε-interface: \IfPropertyExistsTF.
Tests if the ⟨property⟩ has been declared.

\property_if_recorded_p:n ⋆ \property_if_recorded_p:n {⟨label⟩}
\property_if_recorded_p:e ⋆ \property_if_recorded:nTF {⟨label⟩} {⟨true code⟩} {⟨false code⟩}
\property_if_recorded:nTF ⋆
\property_if_recorded:eTF ⋆

LaTeX 2ε-interface: \IfLabelExistsTF
Tests if the ⟨label⟩ is known. This is also true if the label has been set with the standard \label command.

\property_if_recorded_p:nn ⋆ \property_if_recorded_p:nn {⟨label⟩} {⟨property⟩}
\property_if_recorded_p:ee ⋆ \property_if_recorded:nnTF {⟨label⟩} {⟨property⟩} {⟨true code⟩} {⟨false code⟩}
\property_if_recorded:nnTF ⋆
\property_if_recorded:eeTF ⋆

LaTeX 2ε-interface: \IfPropertyRecordedTF.
Tests if the label ⟨label⟩ is known and if it provides a value of the ⟨property⟩.

7 Auxiliary file interfaces

\new@label@record \new@label@record {⟨label⟩} {⟨data⟩}

This is a command only for use in the .aux file. It loads the key–value list of ⟨data⟩ to be available for the ⟨label⟩.

8 LaTeX 2ε interface

The LaTeXe interfaces always expand label and property arguments. This means that one must be careful when using active chars or commands in the names. UTF8-chars are protected and should be safe, similar most babel shorthands.

\NewProperty \NewProperty {⟨property⟩} {⟨setpoint⟩} {⟨default⟩} {⟨code⟩}
\SetProperty \SetProperty {⟨property⟩} {⟨setpoint⟩} {⟨default⟩} {⟨code⟩}

Sets the ⟨property⟩ to have the ⟨default⟩ specified, and at the ⟨setpoint⟩ (either now or shipout) to write the result of the ⟨code⟩ as part of a label. The ⟨code⟩ should be expandable. The expansion of ⟨code⟩ (the value of the property) is written to the .aux file and read back from there at the next compilation (at which point normally the standard LaTeX catcode régime with @ a letter is active).

\RecordProperties \RecordProperties {⟨label⟩} {⟨clist⟩}

Writes the list of properties given by the ⟨clist⟩ to the .aux file with the ⟨label⟩ specified. Similar to the standard \label command the arguments are expanded. So ⟨clist⟩ can be a macro containing a list of properties. Also similar to the standard \label command, the command is surrounded by an \@bsphack/\@esphack pair to preserve spacing.

\RefProperty ⋆ \RefProperty [⟨local default⟩] {⟨label⟩} {⟨property⟩}

Expands to the value of the ⟨property⟩ for the ⟨label⟩, if available, and the default value of the property or – if given – to ⟨local default⟩ otherwise. If {⟨property⟩} has not been declared an error is issued.

`\IfPropertyExistsTF` `\IfPropertyExistsTF {⟨property⟩} {⟨true code⟩} {⟨false code⟩}`
`\IfPropertyExistsT`
`\IfPropertyExistsF`
Tests if the ⟨*property*⟩ has been declared.

`\IfLabelExistsTF` `\IfLabelExistsTF {⟨label⟩} {⟨true code⟩} {⟨false code⟩}`
`\IfLabelExistsT`
`\IfLabelExistsF`
Tests if the ⟨*label*⟩ has been recorded. This is also true if a label has been set with the standard `\label` command.

`\IfPropertyRecordedTF` `\IfPropertyRecordedTF {⟨label⟩} {⟨property⟩} {⟨true code⟩} {⟨false code⟩}`
`\IfPropertyRecordedT`
`\IfPropertyRecordedF`
Tests if the label and a value of the ⟨*property*⟩ for the ⟨*label*⟩ are both known.

`\RefUndefinedWarn` `\RefUndefinedWarn {⟨label⟩} {⟨property⟩}`

Triggers the standard warning

`LaTeX Warning: There were undefined references.`

at the end of the document if the reference for ⟨*label*⟩ and ⟨*property*⟩ can not be resolved. At the point where it is called it also issues the warning

`Reference '⟨label⟩' on page ⟨page⟩ undefined`

if the label is unknown, or the more specific

`Property '⟨property⟩' undefined for reference '⟨label⟩' on page ⟨page⟩` if the label is known but doesn't provide a value for the requested property.

9 Pre-declared properties

`abspage` (shipout) The absolute value of the current page: starts at 1 and increases monotonically at each shipout.

`page` (shipout) The current page as given by `\thepage`: this may or may not be a numerical value, depending on the current style. Contrast with `\abspage`. You get this value also with the standard `\label`/`\pageref`.

`pagenum` (shipout) The current page as arabic number. This is suitable for integer operations and comparisons.

`label` (now) The content of `\@currentlabel`. This is the value that you get also with the standard `\label`/`\ref`.

`title` (now) The content of `\@currentlabelname`. This command is filled beside others by the nameref package and some classes (e.g. memoir).

target (now) The content of `\@currentHref`. This command is normally filled by for example hyperref and gives the name of the last destination it created.

pagetarget (shipout) The content of `\@currentHpage`. This command is filled for example by a recent version of hyperref and then gives the name of the last page destination it created.

counter (now) The content of `\@currentcounter`. This command contains after a `\refstepcounter` the name of the counter.

xpos
ypos (shipout) This stores the x and y coordinates of a point previously stored with `\pdfsavepos`/`\savepos`. E.g. (if bidi is used it can be necessary to save the position before and after the label):

```
\tex_savepos:D
\property_record:nn{myposition}{xpos,ypos}
\tex_savepos:D
```

10 The Implementation

```
⟨*2ekernel | latexrelease⟩
\ExplSyntaxOn
⟨@@=property⟩
⟨latexrelease⟩\NewModuleRelease{2023/11/01}{ltproperties}
⟨latexrelease⟩                        {Cross-referencing~properties}
```

The approach here is based closely on that from zref; separate out lists of properties and the properties themselves, so the latter can be used multiple times and in varying combinations. However, not everything is a straight copy. Firstly, we treat lists of properties as simple comma lists: that allows us to have either saved or dynamic lists and to avoid another data structure. The cost is that errors are detected at point-of-use, but in any real case that should be true anyway (and is true for `\zref@labelbyprop` already). Secondly, we allow properties to have arbitrary names, as the code does not require them to tokenize as control sequences.

\property_new:nnnn
\property_gset:nnnn
__property_gset:nnnn

As properties can be reset, they are not constants. But they also have various pieces of required data. So we use the same approach as color and make them declarations. Data-wise, we need the detail of the implementation, the default and a flag to show if the code works now or at shipout. This last entry is done using text so needs a check. We could use a set of `prop` here, but as we never need to map or copy the lists, we can gain performance using the hash table approach.

```
\cs_new_protected:Npn \property_new:nnnn #1#2#3#4
  {
    \cs_if_free:cTF { __property_code_ #1 : }
      {
        \exp_args:Nx \__property_gset:nnnn { \tl_to_str:n {#1} }
```

```
          {#2} {#3} {#4}
      }
      {
        \msg_error:nn { property }{ exists }{#1}
      }
  }
\cs_new_protected:Npn \property_gset:nnnn #1#2#3#4
  {
    \__property_gset:ennn { \tl_to_str:n {#1} }
      {#2} {#3} {#4}
  }
\cs_new_protected:Npn \__property_gset:nnnn #1#2#3#4
  {
    \cs_gset:cpn { __property_code_ #1 : } {#4}
    \tl_gclear_new:c { g__property_default_ #1 _tl }
    \tl_gset:cn { g__property_default_ #1 _tl } {#3}
    \bool_if_exist:cF { g__property_shipout_ #1 _bool }
      { \bool_new:c { g__property_shipout_ #1 _bool } }
    \str_case:nnF {#2}
      {
        { now } { { \bool_gset_false:c { g__property_shipout_ #1 _bool } } }
        { shipout }
          { \bool_gset_true:c { g__property_shipout_ #1 _bool } }
      }
      { \msg_error:nnnn { property } { unknown-setpoint } {#1} {#2} }
  }
\cs_generate_variant:Nn \__property_gset:nnnn {ennn}
```

(*End of definition for* `\property_new:nnnn`, `\property_gset:nnnn`, *and* `\__property_gset:nnnn`. *These functions are documented on page 3.*)

`\NewProperty`
`\SetProperty`

For consistency we expand the property name, but this doesn't warant a variant of the L3-commands.

```
\cs_new_protected:Npn \NewProperty #1#2#3#4
  {
    \protected@edef\reserved@a{#1}
    \exp_args:No \property_new:nnnn {\reserved@a} {#2}{#3}{#4}
  }
\cs_new_protected:Npn \SetProperty #1#2#3#4
  {
    \protected@edef\reserved@a{#1}
    \exp_args:No \property_gset:nnnn {\reserved@a} {#2}{#3}{#4}
  }
```

(*End of definition for* `\NewProperty` *and* `\SetProperty`. *These functions are documented on page 5.*)

`\property_record:nN`
`\property_record:nn`
`\property_record:nV`
`\property_record:ee`
`\property_record:oo`
`\__property_record:nn`
`\__property_record:en`
`\__property_record_value:n`
`\__property_record_value_aux:n`
`\__property_record_value_aux:e`

Writing data when it is labelled means expanding at this stage and possibly later too. That is all pretty easy using expl3: we accept a stray comma at the end of the list as that is easier to deal with than trying to tidy up, and there is no real downside.

```
\cs_new_protected:Npn \property_record:nN #1#2
  { \property_record:nV {#1} #2 }
\cs_new_protected:Npn \property_record:nn #1#2
  { \__property_record:en { \tl_to_str:n {#1} } {#2} }
\cs_generate_variant:Nn \property_record:nn { nV , ee, oo }
```

```
\cs_new_protected:Npn \__property_record:nn #1#2
  {
    \protected@write \@auxout {}
      {
        \token_to_str:N \new@label@record
          {#1}
          { \clist_map_function:nN {#2} \__property_record_value:n }
      }
  }
\cs_generate_variant:Nn \__property_record:nn { e }
\cs_new:Npn \__property_record_value:n #1
  { \__property_record_value_aux:e { \tl_to_str:n {#1} } }
\cs_new:Npn \__property_record_value_aux:n #1
  {
    \cs_if_exist:cTF { __property_code_ #1 : }
      {
        {#1}
        {
          \bool_if:cTF { g__property_shipout_ #1 _bool }
            { \exp_not:c }
            { \use:c }
              { __property_code_ #1 : }
        }
      }
      { \msg_expandable_error:nnn { property } { not-declared } {#1} }
  }
\cs_generate_variant:Nn \__property_record_value_aux:n { e }
```

(*End of definition for* `\property_record:nN` *and others. These functions are documented on page 3.*)

`\RecordProperties`

```
\NewDocumentCommand\RecordProperties { m m }
  {
    \@bsphack
    \protected@edef\reserved@a{#1}
    \protected@edef\reserved@b{#2}
    \property_record:oo {\reserved@a}{\reserved@b}
    \@esphack
  }
```

(*End of definition for* `\RecordProperties`*. This function is documented on page 5.*)

10.1 Reference commands

`l__property_ref_flag` A flag that is set if a reference couldn't be resolved.

```
\flag_new:n { l__property_ref_flag }
```

(*End of definition for* `l__property_ref_flag`*.*)

`\property_ref:nn` `\property_ref:ee` Search for the label/property combination, and if not found fall back to the default of the property.

```
\cs_new:Npn \property_ref:nn #1#2
  {
    \__property_ref:een
```

```
      { \tl_to_str:n {#1} }
      { \tl_to_str:n {#2} }
      { \tl_use:c { g__property_default_ #2 _tl } }
  }
\cs_generate_variant:Nn \property_ref:nn {ee}
```

(*End of definition for* `\property_ref:nn`. *This function is documented on page* 3.)

`\property_ref:nnn` `\property_ref:een` `\__property_ref:nnn` `\__property_ref:een`

This allows to set a local default value which overrides the default value of the property.

```
\cs_new:Npn \property_ref:nnn #1#2#3
  {
    \__property_ref:een
      { \tl_to_str:n {#1} }
      { \tl_to_str:n {#2} }
      {#3}
  }
\cs_new:Npn \__property_ref:nnn #1#2#3
  {
    \tl_if_exist:cTF { g__property_label_ #1 _ #2 _tl }
      { \tl_use:c { g__property_label_ #1 _ #2 _tl } }
      {
        \flag_if_raised:nF
          { l__property_ref_flag } { \flag_raise:n { l__property_ref_flag } }
```

We test for the default of the property only to check if the property has been declared.

```
        \tl_if_exist:cTF { g__property_default_ #2 _tl }
          { #3 }
          { \msg_expandable_error:nnn { property } { not-declared } {#2} }
      }
  }
\cs_generate_variant:Nn \__property_ref:nnn { ee }
\cs_generate_variant:Nn \property_ref:nnn {een}
```

(*End of definition for* `\property_ref:nnn` *and* `\__property_ref:nnn`. *This function is documented on page* 4.)

`\property_item:nn` `\property_item:ee`

Retrieve the value but wrap it into an `\exp_not:n` to avoid further expansion.

```
\cs_new:Npn \property_item:nn #1#2
 {
  \tl_if_exist:cTF { g__property_label_ \tl_to_str:n {#1} _ \tl_to_str:n {#2} _tl }
   {
     \exp_not:v { g__property_label_ \tl_to_str:n {#1} _ \tl_to_str:n {#2} _tl }
   }
   { \tl_use:c { g__property_default_ \tl_to_str:n {#2} _tl } }
 }
\cs_generate_variant:Nn\property_item:nn{ee}
```

(*End of definition for* `\property_item:nn`. *This function is documented on page* 4.)

`\RefProperty`

Search for the label/property combination, and if not found fall back to the default of the property or the given default.

```
\NewExpandableDocumentCommand \RefProperty { o m m }
  {
    \IfNoValueTF {#1}
      {
```

```
      \property_ref:ee {#2}{#3}
    }
    {
      \property_ref:een {#2}{#3}{#1}
    }
}
```

(*End of definition for* `\RefProperty`. *This function is documented on page 5.*)

`\new@label@record`
`\__property_data:nnn`

A standard recursion loop.

```
\cs_new_protected:Npn \new@label@record #1#2
  {
    \tl_if_exist:cTF { r@#1 }
      {
        \gdef \@multiplelabels
          { \@latex@warning@no@line { There~were~multiply-defined~labels } }
        \@latex@warning@no@line { Label~`#1'~multiply~defined }
      }
      {
        \tl_new:c { r@#1 }
        \tl_gset:cn { r@#1 }{#2}
      }
    \__property_data:nnn {#1} #2 { \q_recursion_tail } { ? } \q_recursion_stop
  }
\cs_new_protected:Npn \__property_data:nnn #1#2#3
  {
    \quark_if_recursion_tail_stop:n {#2}
    \tl_gclear_new:c { g__property_label_ \tl_to_str:n {#1} _ \tl_to_str:n {#2} _tl }
    \tl_gset:cn { g__property_label_ \tl_to_str:n {#1} _ \tl_to_str:n {#2} _tl } {#3}
    \__property_data:nnn {#1}
  }
```

This command is used in `\enddocument` to test if some label values have changed.

```
\cs_new_protected:Npn \@kernel@new@label@record@testdef #1 #2
  {
    \tl_if_eq:cnF { r@#1 } {#2}
      { \@tempswatrue }
  }
```

(*End of definition for* `\new@label@record` *and* `\__property_data:nnn`. *This function is documented on page 5.*)

10.2 Tests and warnings

`\property_if_exist_p:n`
`\property_if_exist:nTF`

Tests if property has been declared.

```
\prg_new_conditional:Npnn \property_if_exist:n #1 { p , T , F,  TF }
  % #1 property
  {
    \cs_if_exist:cTF { __property_code_ #1 : }
      {
        \prg_return_true:
      }
      {
        \prg_return_false:
      }
```

```
  }
\prg_generate_conditional_variant:Nnn \property_if_exist:n {e} { p , T , F,  TF }
```

(*End of definition for* `\property_if_exist:nTF`. *This function is documented on page* 4.)

`\IfPropertyExistsTF`
`\IfPropertyExistsT`
`\IfPropertyExistsF`

```
\cs_new_eq:NN \IfPropertyExistsTF \property_if_exist:eTF
\cs_new:Npn   \IfPropertyExistsT #1#2 {\property_if_exist:eTF {#1}{#2}{} }
\cs_new:Npn   \IfPropertyExistsF #1   {\property_if_exist:eTF {#1}{} }
```

(*End of definition for* `\IfPropertyExistsTF`, `\IfPropertyExistsT`, *and* `\IfPropertyExistsF`. *These functions are documented on page* 6.)

`\property_if_recorded_p:n`
`\property_if_recorded:nTF`

Tests if the label has been set. This can then be used to setup e.g. rerun messages.

```
\prg_new_conditional:Npnn \property_if_recorded:n #1 { p , T , F,  TF }
  % #1 label
  {
    \tl_if_exist:cTF { r@#1 }
      {
        \prg_return_true:
      }
      {
        \prg_return_false:
      }
  }
\prg_generate_conditional_variant:Nnn \property_if_recorded:n {e} { p , T , F,  TF }
```

(*End of definition for* `\property_if_recorded:nTF`. *This function is documented on page* 5.)

`\IfLabelExistsTF`
`\IfLabelExistsT`
`\IfLabelExistsF`

```
\cs_new_eq:NN \IfLabelExistsTF \property_if_recorded:eTF
\cs_new:Npn   \IfLabelExistsT #1#2 {\property_if_recorded:eTF {#1}{#2}{} }
\cs_new:Npn   \IfLabelExistsF #1   {\property_if_recorded:eTF {#1}{} }
```

(*End of definition for* `\IfLabelExistsTF`, `\IfLabelExistsT`, *and* `\IfLabelExistsF`. *These functions are documented on page* 6.)

`\property_if_recorded_p:nn`
`\property_if_recorded:nnTF`

tests if the label/property combination has been set This can then be used to setup e.g. rerun messages.

```
\prg_new_conditional:Npnn \property_if_recorded:nn #1#2 { p , T , F,  TF }
  % #1 label #2 property
  {
    \tl_if_exist:cTF { g__property_label_ \tl_to_str:n {#1} _ \tl_to_str:n {#2} _tl }
      {
        \prg_return_true:
      }
      {
        \prg_return_false:
      }
  }
\prg_generate_conditional_variant:Nnn \property_if_recorded:nn {ee} { p , T , F,  TF }
```

(*End of definition for* `\property_if_recorded:nnTF`. *This function is documented on page* 5.)

\IfPropertyRecordedTF
\IfPropertyRecordedT
\IfPropertyRecordedF

```
\cs_new_eq:NN \IfPropertyRecordedTF \property_if_recorded:eeTF
\cs_new:Npn \IfPropertyRecordedT #1#2#3 { \property_if_recorded:eeTF {#1}{#2}{#3}{} }
\cs_new:Npn \IfPropertyRecordedF #1#2#3 { \property_if_recorded:eeTF {#1}{#2}{}{#3} }
```

(*End of definition for* `\IfPropertyRecordedTF`, `\IfPropertyRecordedT`, *and* `\IfPropertyRecordedF`. *These functions are documented on page* 6.)

\property_ref_undefined_warn:

`\G@refundefinedtrue` is defined in `ltxref` and redefines a warning message.

```
\cs_new_protected:Npn \property_ref_undefined_warn:
 {
   \flag_if_raised:nT { l__property_ref_flag }
    {
      \G@refundefinedtrue
    }
 }
```

(*End of definition for* `\property_ref_undefined_warn:`. *This function is documented on page* 4.)

\property_ref_undefined_warn:n
\property_ref_undefined_warn:e

```
\cs_new_protected:Npn \property_ref_undefined_warn:n #1 %#1 label
  {
   \property_if_recorded:nF {#1}
    {
       \G@refundefinedtrue
       \@latex@warning{Reference~‘#1’~on~page~\thepage\space undefined}%
    }
  }
\cs_generate_variant:Nn \property_ref_undefined_warn:n {e}
```

(*End of definition for* `\property_ref_undefined_warn:n`. *This function is documented on page* 4.)

\property_ref_undefined_warn:nn
\property_ref_undefined_warn:ee
\RefUndefinedWarn

```
\cs_new_protected:Npn \property_ref_undefined_warn:nn #1#2 %#1 label, #2 property
 {
   \property_if_recorded:nTF {#1}
    {
      \property_if_recorded:nnF {#1}{#2}
        {
          \G@refundefinedtrue
          \@latex@warning
           { Property~‘#2’~undefined~for~reference~‘#1’~on~page~\thepage }
        }
    }
    {
       \G@refundefinedtrue
       \@latex@warning { Reference~‘#1’~on~page~\thepage\space undefined }%
    }
 }
\cs_generate_variant:Nn \property_ref_undefined_warn:nn {ee}
\cs_set_eq:NN \RefUndefinedWarn \property_ref_undefined_warn:ee
```

(*End of definition for* `\property_ref_undefined_warn:nn` *and* `\RefUndefinedWarn`. *These functions are documented on page* 4.)

10.3 Predeclared properties

abspage

```
\property_new:nnnn { abspage } { shipout }
  { 0 } { \int_use:N \g_shipout_readonly_int }
```

(*End of definition for* `abspage`. *This variable is documented on page* 6.)

page

```
\property_new:nnnn  { page } { shipout } { 0 } { \thepage }
```

(*End of definition for* `page`. *This variable is documented on page* 6.)

pagenum

```
\property_new:nnnn  { pagenum } { shipout } { 0 } { \the \value { page } }
```

(*End of definition for* `pagenum`. *This variable is documented on page* 6.)

label

```
\property_new:nnnn  { label } { now } { ?? } { \@currentlabel }
```

(*End of definition for* `label`. *This variable is documented on page* 6.)

title

```
\property_new:nnnn  { title } { now }
  { \exp_not:n { \textbf { ?? } } } { \@currentlabelname }
```

(*End of definition for* `title`. *This variable is documented on page* 6.)

target

```
\property_new:nnnn  { target } { now } { } { \@currentHref }
```

(*End of definition for* `target`. *This variable is documented on page* 7.)

pagetarget

```
\newcommand\@currentHpage{}
\property_new:nnnn  { pagetarget } { shipout } { } { \@currentHpage }
```

(*End of definition for* `pagetarget`. *This variable is documented on page* 7.)

counter

```
\property_new:nnnn  { counter } { now }  { } { \@currentcounter }
```

(*End of definition for* `counter`. *This variable is documented on page* 7.)

xpos
ypos

```
\property_new:nnnn  { xpos } { shipout } { 0} { \int_use:N \tex_lastxpos:D }
\property_new:nnnn  { ypos } { shipout } { 0} { \int_use:N \tex_lastypos:D }
```

(*End of definition for* `xpos` *and* `ypos`. *These variables are documented on page* 7.)

10.4 Messages

```
\msg_new:nnnn { property } { exists }
    { Property~'#1'~ has~ already~ been~ declared. }
    { There~ already~ exists~ a~ property~ declaration~ with~ this~
      name.\\
      Please~ use~ a~ different~ name~ for~ your~ property.}

\msg_new:nnnn { property } { not-declared }
  { Property~'#1'~not~declared. }
  {
    LaTeX~has~been~asked~to~use~property~'#1',~but~this~
    name~has~not~been~declared.
  }
\msg_new:nnnn { property } { unknown-setpoint }
  { Unknown~keyword~'#2'~for~setting~property~'#1'. }
  {
    LaTeX~has~been~asked~to~set~the~property~'#1',~but~the~keyword~
    '#2'~is~not~one~of~the~two~known~values:~'now'~or~'shipout'.
  }
%
⟨latexrelease⟩\IncludeInRelease{0000/00/00}{ltproperties}
⟨latexrelease⟩                  {cross-referencing~properties~(undo)}%
⟨latexrelease⟩
⟨latexrelease⟩\let \NewProperty  \@undefined
⟨latexrelease⟩\let \SetProperty \@undefined
⟨latexrelease⟩
⟨latexrelease⟩\let \RecordProperties  \@undefined
⟨latexrelease⟩\let \RefProperty \@undefined
⟨latexrelease⟩\let \RefUndefinedWarn  \@undefined
⟨latexrelease⟩
⟨latexrelease⟩\let \IfPropertyExistsTF \@undefined
⟨latexrelease⟩\let \IfLabelExistsTF \@undefined
⟨latexrelease⟩\let \IfPropertyRecordedTF  \@undefined
⟨latexrelease⟩
⟨latexrelease⟩\let\new@label@record \@undefined
⟨latexrelease⟩\let\@kernel@new@label@record@testdef\@undefined
⟨latexrelease⟩\EndModuleRelease

\ExplSyntaxOff

⟨/2ekernel | latexrelease⟩
```

Reset module prefix:

```
⟨@@=⟩
```