

## 1 Package header

```
1 \*package>
2 \@@=yoin>
```

Necessary packages: First, L<sup>A</sup>T<sub>E</sub>X3 stuff.

```
3 \RequirePackage{expl3,l3keys2e,l3regex,xparse}
```

From zref bundle, for computing the total number of pages of an article.

```
4 \RequirePackage{zref-totpages}
```

We need the absolute paths. This also means we need `-recorder` option to `pdflatex`.

```
5 \RequirePackage[abspath]{currfile}
```

For including PDF files.

```
6 \RequirePackage{pdfpages}
```

Package header.

```
7 \ProvidesExplPackage{yoin}{2016/02/28}{v0.0.1}{Joining articles into issues}
```

## 2 General macros

Macros not necessarily related to the package; moreorless an addition to L<sup>A</sup>T<sub>E</sub>X3.

`\yoin_seq_gappend_clist:Nn` Globally append `clist` #2 to `seq` #1.

```
8 \seq_new:N \l__yoin_seq_tmpa_seq
9 \cs_new_protected:Nn \yoin_seq_gappend_clist:Nn {
10   \seq_set_from_clist:Nn \l__yoin_seq_tmpa_seq { #2 }
11   \seq_gconcat:NNN #1 #1 \l__yoin_seq_tmpa_seq
12 }
```

`\yoin_keys_set_from_file:nn` Read a file #2 containing a key–value list and set the keys for #1. No checks are done here, nothing like comments could be used, the keys should be separated by a comma (and spaces of course as needed).

```
13 \tl_new:N \l__yoin_keys_tmpa_tl
14 \cs_generate_variant:Nn \keys_set:nn { nV }
15 \cs_new_protected:Nn \yoin_keys_set_from_file:nn {
16   \tl_set_from_file:Nnn \l__yoin_keys_tmpa_tl { } { #2 }
17   \keys_set:nV { #1 } \l__yoin_keys_tmpa_tl
18 }
```

`\yoin_keyval_parse_from_file:nn` Read a file #2 containing a key–value list and set the keys for #1. No checks are done here, nothing like comments could be used, the keys should be separated by a comma (and spaces of course as needed).

```
19 \cs_generate_variant:Nn \keyval_parse:NNn { NNv }
20 \cs_new_protected:Nn \yoin_keyval_parse_from_file:NNn {
```

```

21  \tl_set_from_file:Nnn \l__yoin_keys_tmpa_tl { } { #3 }
22  \keyval_parse:NNV #1 #2 \l__yoin_keys_tmpa_tl
23 }

```

msg: boolean-values-only Message for a non-boolean passed to a bool key.

```

24 \msg_new:nnn { yoin } { boolean-values-only }
25   { Key ~ '#1' ~ accepts ~ boolean ~ values ~ only.}

```

### 3 Key-value interface for the package setup

First, we define the variables to store the keys.

```

\g_yoin_subprocess_bool  Booleans:
  \g_yoin_article_bool  26 \bool_new:N \g_yoin_subprocess_bool
  \g_yoin_dryrun_bool    27 \bool_new:N \g_yoin_article_bool
\g_yoin_onlyflags_bool   28 \bool_new:N \g_yoin_dryrun_bool
  \g_yoin_onlytags_bool  29 \bool_new:N \g_yoin_onlyflags_bool
                        30 \bool_new:N \g_yoin_onlytags_bool

```

```

  \g_yoin_flags_seq      Sequences for flags, tags and their filtering:
  \g_yoin_tags_seq       31 \seq_new:N \g_yoin_flags_seq
\g_yoin_onlyflags_seq    32 \seq_new:N \g_yoin_tags_seq
  \g_yoin_onlytags_seq   33 \seq_new:N \g_yoin_onlyflags_seq
                        34 \seq_new:N \g_yoin_onlytags_seq

```

\g\_yoin\_jobname\_tl We can modify what the package considers as the value of \jobname, here's a token list for that:

```

35 \tl_new:N \g_yoin_jobname_tl
36 \tl_gset_eq:NN \g_yoin_jobname_tl \c_job_name_tl

```

msg: unknown-flag Two messages, for unknown flags and unknown tags.

```

msg: unknown-tag 37 \msg_new:nnnn { yoin } { unknown-flag }
                  38   { The ~ flag ~ '#1' ~ is ~ unknown ~ to ~ 'yoin'. }
                  39   { You ~ either ~ misspelled ~ it ~ or ~forgot ~ to ~ declare ~ it. }

                  40 \msg_new:nnnn { yoin } { unknown-tag }
                  41   { The ~ tag ~ '#1' ~ is ~ unknown ~ to ~ 'yoin'. }
                  42   { You ~ either ~ misspelled ~ it ~ or ~forgot ~ to ~ declare ~ it. }

```

```

\yoin_if_tag_defined:n Conditionals for checking whether a tag/flag was defined.
\yoin_if_flag_defined:n 43 \prg_new_protected_conditional:Nnn \yoin_if_tag_defined:n { T, F, TF } {
44   \seq_if_in:NnTF \g_yoin_tags_seq { #1 } { \prg_return_true: } { \prg_return_false: }
45 }

46 \prg_new_protected_conditional:Nnn \yoin_if_flag_defined:n { T, F, TF } {
47   \seq_if_in:NnTF \g_yoin_flags_seq { #1 } { \prg_return_true: } { \prg_return_false: }
48 }

```

```

\_yoin_error_if_tag_undefined:n Check whether a tag/flag is defined, if not, issue an error.
\_yoin_error_if_flag_undefined:n 49 \cs_new_protected:Nn \_yoin_error_if_tag_undefined:n {
50   \yoin_if_tag_defined:nF { #1 } { \msg_error:nnn { yoin } { unknown-tag } { #1 } }
51 }

52 \cs_new_protected:Nn \_yoin_error_if_flag_undefined:n {
53   \yoin_if_flag_defined:nF { #1 } { \msg_error:nnn { yoin } { unknown-flag } { #1 } }
54 }

```

yoin / general The keys themselves:

```

55 \keys_define:nn { yoin / general } {
Booleans:
56   dryrun .bool_gset:N = \g_yoin_dryrun_bool,
57   dryrun .initial:n = { false },

58   article .bool_gset:N = \g_yoin_article_bool,
59   article .initial:n = { false },

60   subprocess .bool_gset:N = \g_yoin_subprocess_bool,
61   subprocess .initial:n = { false },

Keys whose clist values are appended to a seq:
62   defineflags .code:n = \yoin_seq_gappend_clist:Nn \g_yoin_flags_seq { #1 },

63   definetags .code:n = \yoin_seq_gappend_clist:Nn \g_yoin_tags_seq { #1 },

```

A clist key is stored in a seq, also, a corresponding bool is set true. (The point is, if onlyflags/onlytags is not ever set up, we want to know it since we treat it as if we use all flags/tags.)

```

64   onlyflags .code:n =
65     \seq_gset_from_clist:Nn \g_yoin_onlyflags_seq { #1 }
66     \bool_gset_true:N \g_yoin_onlyflags_bool
67   ,

```

```

68   onlytags .code:n =
69     \seq_gset_from_clist:Nn \g_yoin_onlytags_seq { #1 }
70     \bool_gset_true:N \g_yoin_onlytags_bool
71     ,

```

A key whose value is stored in a token list.

```

72   jobname .tl_gset:N = \g_yoin_jobname_tl,

73 % A key that allows |\yoinMeta| to be called from within the package options.
74 %   \begin{macrocode}
75   meta .code:n = \yoin_yoinmeta:n { #1 },

76 }

```

`\ProcessKeysPackageOptions` Process key options given to the package. We *do not want to process any options given to the class*. Whence `\ProcessKeysPackageOptions` and not `\ProcessKeysOptions`.

```

77 \ProcessKeysPackageOptions { yoin / general }

```

`\yoin_setup:n` Allow keys to be set later. We define both a  $\text{\LaTeX}$ 3 interface and an xparse UI wrapper.

```

\yoinSetup
78 \cs_new_protected:Nn \yoin_setup:n {
79   \keys_set:nn { yoin / general } { #1 }
80 }

81 \NewDocumentCommand \yoinSetup { R[]{} } {
82   \yoin_setup:n { #1 }
83 }

```

## 4 yoinMeta macro — adding issue's

```

\yoinMeta
\yoin_yoinmeta:n
84 \prop_new:N \l__yoin_yoinmeta_prop
85 \cs_new_protected:Nn \__yoin_yoinmeta_storekey:nn {
86   \prop_put:Nnn \l__yoin_yoinmeta_prop { #1 } { #2 }
87 }
88 \cs_new_protected:Nn \__yoin_yoinmeta_storekey:n {
89   \prop_put:Nnn \l__yoin_yoinmeta_prop { #1 } { }
90 }
91 \cs_new_protected:Nn \yoin_yoinmeta:n {
92   \keyval_parse:NNn \__yoin_yoinmeta_storekey:n \__yoin_yoinmeta_storekey:nn { #1 }

```

```

93 }
94 \NewDocumentCommand \yoinMeta { R[]{} } {
95   \yoin_yoinmeta:n { #1 }
96 }

```

## 5 yoinAdd macro — adding articles to the issue

The key–value interface. In this case, we basically only store the keys for each article in a prop. First, an interface for setting the keys for the articles. `\yoin_yoinadd_prop:n` returns the name of the prop for the given article; *no check for existence is done at this place*.

```

\g_yoin_yoinadd_seq A sequence for storing the list of the existing articles.
97 \seq_new:N \g_yoin_yoinadd_seq

\yoin_yoinadd_prop:n \yoin_yoinadd_prop:n returns the name of the prop for the given article; no check for existence is done at this place.
\yoin_yoinadd_prop:V \yoin_yoinadd_prop:nn returns property \#2 of article \#1, or \q_no_value if the property is not set.
\yoin_yoinadd_prop_item:nn 98 \cs_new:Nn \yoin_yoinadd_prop:n {
\yoin_yoinadd_prop_item:Vn 99   g__yoin_article_#1_prop
100 }
101 \cs_generate_variant:Nn \yoin_yoinadd_prop:n { V }

102 \cs_new:Nn \yoin_yoinadd_prop_item:nn {
103   \prop_item:cn { \yoin_yoinadd_prop:n { #1 } } { #2 }
104 }
105 \cs_generate_variant:Nn \yoin_yoinadd_prop_item:nn { V }

```

For processing `\yoinAdd`, we first set up a `tl` to contain the name of the article, then create the prop, and finally use `l3keys` to fill in the prop. Note that if an article is added twice, an error is issued, if the error is ignored, the article is not added but the properties are set.

```

\l__yoin_yoinadd_currentarticle_tl A tl that stores the name of the article that is being processed by \yoinAdd.
106 \tl_new:N \l__yoin_yoinadd_currentarticle_tl

__yoin_yoinadd_storekey:nn Internal macro for storing a key in the prop. The one-parameter variant sets the value of the key empty.
__yoin_yoinadd_storekey:n 107 \cs_new_protected:Nn \__yoin_yoinadd_storekey:nn {
108   \prop_gput:cnn { \yoin_yoinadd_prop:V \l__yoin_yoinadd_currentarticle_tl } { #1 } { #2 }
109 }
110 \cs_new_protected:Nn \__yoin_yoinadd_storekey:n {
111   \prop_gput:cnn { \yoin_yoinadd_prop:V \l__yoin_yoinadd_currentarticle_tl } { #1 } { }
112 }

```

`\yoin_yoinadd:nn` The macro `\yoinAdd` itself. We first set `\l_@@_yoinadd_currentarticle_tl`, then check whether the same article has not been processed before (issuing an error in that case and finishing). Then, the article is added in `\g_yoin_yoinadd_seq`, the prop created, the article's name added in the prop with key `article` and the keys are set. If the article has a `.yoin` file in its sub-directory, the key-values in it is added to the prop. If the file does not exist, it means things are wrong (the article should first be set up, before being added to its issue by `\yoinAdd`).

```

113 \cs_new_protected:Nn \yoin_yoinadd:nn {
114   \tl_set:Nn \l__yoin_yoinadd_currentarticle_tl { #1 }
115   \seq_if_in:NnTF \g_yoin_yoinadd_seq { #1 } {
116     \msg_error:nnn { yoin } { yoinadd-duplicatearticle } { #1 }
117   } {
118     \seq_gput_right:Nn \g_yoin_yoinadd_seq { #1 }
119     \prop_new:c { \yoin_yoinadd_prop:n { #1 } }
120     \clist_map_inline:nn { forceopenany, forceopenright, ignore } {
121       \__yoin_yoinadd_storekey:nn { ##1 } { 0 }
122     }
123     \__yoin_yoinadd_storekey:nn { article } { #1 }
124     \keys_set:nn { yoin / yoinadd } { #2 }
125     \file_if_exist:nTF { #1 / #1 .yoin } {
126       \yoin_keyval_parse_from_file:NNn
127         \__yoin_yoinadd_storekey:n
128         \__yoin_yoinadd_storekey:nn
129         { #1 / #1 .yoin }
130     } {
131       \msg_error:nnn { yoin } { yoinadd-dotyoinmissing } { #1 }
132     }
133   }
134 }

135 \NewDocumentCommand \yoinAdd { m O{} } {
136   \yoin_yoinadd:nn { #1 } { #2 }
137 }

```

`msg: yoinadd-duplicatearticle` The error messages: for adding a duplicate article and for adding an article with no `#1/#1.yoin` file.

```

msg: yoinadd-dotyoinmissing 138 \msg_new:nnn { yoin } { yoinadd-duplicatearticle }
                             139   { The ~ article ~ "#1" ~ has ~ been ~ already ~ processed ~ by ~ \token_to_str:N \yoinAdd ~.}
                             140 \msg_new:nnn { yoin } { yoinadd-dotyoinmissing }
                             141   { The ~ article ~ "#1" ~ has ~ no ~ file ~ "#1/#1.yoin" ~ and ~ was ~ not ~ properly ~ set ~ up.}

```

`yoin / yoinadd` The keys here are pretty simple; each defined key just stores its value in the prop. We recall that `\#1` is the key and `\#\#1` is the value.

```

142 \clist_map_inline:nn { textualkey } {
143   \keys_define:nn { yoin / yoinadd } {
144     #1 .code:n = \__yoin_yoinadd_storekey:nn { #1 } { ##1 },
145   }
146 }

```

For boolean keys, we create a manual boolean parser.

```

147 \clist_map_inline:nn { forceopenany, forceopenright, ignore } {
148   \keys_define:nn { yoin / yoinadd } {
149     #1 .choice:,
150     #1 / true .code:n = \__yoin_yoinadd_storekey:nn { #1 } { 1 },
151     #1 / false .code:n = \__yoin_yoinadd_storekey:nn { #1 } { 0 },
152     #1 / unknown .code:n = \msg_error:nnx { yoin } { boolean-values-only } { \l_keys_key_tl },
153   }
154 }

```

However, for the tag key, we additionally check that the tag exists.

```

155 \keys_define:nn { yoin / yoinadd } {
156   tag .code:n =
157     \__yoin_error_if_tag_undefined:n { #1 }
158     \__yoin_yoinadd_storekey:nn { tag } { #1 }
159   ,
160 }

```

## 6 Environment yoinshell

`\l_yoin_yoinshell_ignore_bool` A boolean for storing the ignore key's value.

`yoin / yoinshell` Key-value interface to yoinshell.

```

161 \keys_define:nn { yoin / yoinshell } {

```

If flag is set and onlyflags is set but the flag is not amongst them, the whole yoinshell is ignored (by setting the ignore key).

```

162   flag .code:n =
163     \__yoin_error_if_flag_undefined:n { #1 }
164     \bool_if:NT \g_yoin_onlyflags_bool {
165       \seq_if_in:NnF \g_yoin_onlyflags_seq { #1 } {
166         \keys_set:nn { yoin / yoinshell } {
167           ignore = true
168         }
169       }

```

```

170     }
171     ,
    The ignore key sets a boolean
172     ignore .bool_set:N = \l_yoin_yoinshell_ignore_bool,
173     ignore .initial:n = { false },
174 }

```

`shellesc.sty` A reasonable shell escape that should work in both pdf<sub>l</sub>atex and lua<sub>l</sub>atex in T<sub>E</sub>X Live 2016.

```

\ShellEscape 175 \file_if_exist:nTF { shellesc.sty } {
\__yoin_yoinshell_shellescape:n 176   \RequirePackage { shellesc }
177 } {
178   \def \ShellEscape #1 { \immediate \write 18 { #1 } }
179 }
180 \cs_new_protected:Nn \__yoin_yoinshell_shellescape:n {
181   \ShellEscape { #1 }
182 }

```

`\__yoin_yoinshell_begin:n` Environment yoinshell (one key–value argument). We perform some local definitions that should stay local, so we put everything in a group. The keys are set. Then we define the macros — “shell commands”. If ignore is set or if subprocess is passed to the package, these macros are declared to do nothing, otherwise they are simply wrappers to the L<sup>A</sup>T<sub>E</sub>X3 counterparts.

```

183 \cs_new_protected:Nn \__yoin_yoinshell_begin:n {
184   \group_begin:
185   \keys_set:nn { yoin / yoinshell } { #1 }
186   \bool_if:NT \g_yoin_subprocess_bool {
187     \bool_set_true:N \l_yoin_yoinshell_ignore_bool
188   }
189   \bool_if:NTF \l_yoin_yoinshell_ignore_bool {
190     \DeclareDocumentCommand \RunForEach { 0{} m } { }
191     \DeclareDocumentCommand \Run { 0{} m } { }
192     \DeclareDocumentCommand \WriteMeta { 0{} } { }
193   } {
194     \DeclareDocumentCommand \RunForEach { 0{} m } { \yoin_yoinshell_runforeach:nn { ##1 } { ##2 } }
195     \DeclareDocumentCommand \Run { 0{} m } { \yoin_yoinshell_run:nn { ##1 } { ##2 } }
196     \DeclareDocumentCommand \WriteMeta { 0{} } { \yoin_yoinshell_writemeta:n { ##1 } }
197     \yoin_yoinshell_writemeta:n { }
198   }
199 }

```



```

200 \cs_new_protected:Nn \__yoin_yoinshell_end: {
201   \group_end:
202 }

203 \NewDocumentEnvironment { yoinshell } { 0{ } } {
204   \__yoin_yoinshell_begin:n { #1 }
205 } {
206   \__yoin_yoinshell_end:
207 }

```

## 6.1 RunForEach

\l\_\_yoin\_yoinshell\_runforarticle\_tag\_tl First, two tls that will store tags: One for the tag of the article, one that could be passed to \RunForEach that is initially set to \l\_\_yoin\_yoinshell\_runforeach\_onlytag\_tl \q\_no\_value.

```

208 \tl_new:N \l__yoin_yoinshell_runforarticle_tag_tl
209 \tl_new:N \l__yoin_yoinshell_runforeach_onlytag_tl
210 \tl_set:Nn \l__yoin_yoinshell_runforeach_onlytag_tl { \q_no_value }

```

**yoin / runforeach** So far, the only key–val passable to \RunForEach is onlytag, which tests for the tag to be declared and passes it to \l\_\_@\_yoinshell\_runforeach\_only

```

211 \keys_define:nn { yoin / runforeach } {
212   onlytag .code:n =
213     \__yoin_error_if_tag_undefined:n { #1 }
214     \tl_set:Nn \l__yoin_yoinshell_runforeach_onlytag_tl { #1 }
215   ,
216 }

```

\\_\_yoin\_yoinshell\_runforarticle\_keyfromprop:nnN This macro lets #3 to the value of property #2 of article #1. It makes it an empty definition if the property is unset.

```

217 \tl_new:N \l__yoin_yoinshell_runforarticle_tmpa_tl
218 \cs_new_protected:Nn \__yoin_yoinshell_runforarticle_keyfromprop:nnN {
219   \prop_get:cnN { \yoin_yoinadd_prop:n { #1 } } { #2 } \l__yoin_yoinshell_runforarticle_tmpa_tl
220   \quark_if_no_value:NTF \l__yoin_yoinshell_runforarticle_tmpa_tl {
221     \def #3 {}
222   } {
223     \let #3 \l__yoin_yoinshell_runforarticle_tmpa_tl
224   }
225 }

```

\\_yoin\_yoinshell\_runforeach:nn \RunForEach itself just sets the keys (in a group to make things local) and then calls \@@\_yoinshell\_runforarticle:nn on each article.

```

226 \cs_new_protected:Nn \yoin_yoinshell_runforeach:nn {
227   \group_begin:
228   \keys_set:nn { yoin / runforeach } { #1 }
229   \seq_map_inline:Nn \g_yoin_yoinadd_seq { \_yoin_yoinshell_runforarticle:nn { ##1 } { #2 } }
230   \group_end:
231 }

```

\\_yoin\_yoinshell\_runforarticle:nn If the tag passed to onlytag of \RunForEach is identical to the tag of the article or if any of them is not set, we do what should be done, otherwise nothing is done (the tags do not match). We only extract the prop to publically available macros like \Article, \Jobname etc. (in a group to make this local), and then run the command in shell escape.

```

232 \cs_new_protected:Nn \_yoin_yoinshell_runforarticle:nn {
233   \prop_get:cnN { \yoin_yoinadd_prop:n { #1 } } { tag } \l__yoin_yoinshell_runforarticle_tag_tl
234   \bool_if:nT {
235     \quark_if_no_value_p:N \l__yoin_yoinshell_runforarticle_tag_tl
236     ||
237     \quark_if_no_value_p:N \l__yoin_yoinshell_runforeach_onlytag_tl
238     ||
239     \tl_if_eq_p:NN \l__yoin_yoinshell_runforeach_onlytag_tl \l__yoin_yoinshell_runforarticle_tag_tl
240   }{
241     \group_begin:
242     \_yoin_yoinshell_runforarticle_keyfromprop:nnN { #1 } { article } \Article
243     \_yoin_yoinshell_runforarticle_keyfromprop:nnN { #1 } { jobname } \Jobname
244     \_yoin_yoinshell_runforarticle_keyfromprop:nnN { #1 } { firstpage } \FirstPage
245     \_yoin_yoinshell_shellescape:n { #2 }
246     \group_end:
247   }
248 }

```

## 6.2 Run

\\_yoin\_yoinshell\_run:nn

```

249 \cs_new_protected:Nn \yoin_yoinshell_run:nn {
250   \group_begin:
251   \keys_set:nn { yoin / run } { #1 }
252   \let \Jobname \c_job_name_tl
253   \_yoin_yoinshell_shellescape:n { #2 }
254   \group_end:

```

```
255 }
```

### 6.3 WriteMeta

```
\_yoin_yoinshell_writemeta:n
256 \iow_new:N \g__yoin_yoinshell_iow
257 \cs_new_protected:Nn \yoin_yoinshell_writemeta:n {
258   \group_begin:
259   \yoin_yoinmeta:n { #1 }
260   \iow_open:Nn \g__yoin_yoinshell_iow { \g_yoin_jobname_tl .yoin }
261   \prop_map_inline:Nn \l__yoin_yoinmeta_prop {
262     \iow_now:Nn \g__yoin_yoinshell_iow { meta-##1 ~ = ~ ##2, }
263   }
264   \iow_close:N \g__yoin_yoinshell_iow
265   \group_end:
266 }
```

## 7 macro yoinForEach

`\l__yoin_yoinforeach_article_tag_tl` First, two tls that will store tags: One for the tag of the article, one that could be passed to `\yoinForEach` that is initially set to `\q_no_value`.  
`\l__yoin_yoinforeach_onlytag_tl`

```
267 \tl_new:N \l__yoin_yoinforeach_article_tag_tl
268 \tl_new:N \l__yoin_yoinforeach_onlytag_tl
269 \tl_set:Nn \l__yoin_yoinforeach_onlytag_tl { \q_no_value }
```

`yoin / yoinforeach` So far, the only key-val passable to `\yoinForEach` is `onlytag`, which tests for the tag to be declared and passes it to `\l_@@_yoinforeach_onlytag_tl`.

```
270 \keys_define:nn { yoin / yoinforeach } {
271   onlytag .code:n =
272     \__yoin_error_if_tag_undefined:n { #1 }
273     \tl_set:Nn \l__yoin_yoinforeach_onlytag_tl { #1 }
274   ,
275 }
```

`\_yoin_yoinforeach_article_keyfromprop:nnN` This macro lets #3 to the value of property #2 of article #1. It makes it an empty definition if the property is unset.

```
276 \tl_new:N \l__yoin_yoinforeach_tmpa_tl
277 \cs_new_protected:Nn \_yoin_yoinforeach_article_keyfromprop:nnN {
278   \prop_get:cnN { \yoin_yoinadd_prop:n { #1 } } { #2 } \l__yoin_yoinforeach_tmpa_tl
```

```

279 \quark_if_no_value:NTF \l__yoin_yoinforeach_article_tmpa_tl {
280   \def #3 {(nic)}
281 } {
282   \let #3 \l__yoin_yoinforeach_tmpa_tl
283 }
284 }

```

`\__yoin_yoinforeach_article_metaitem:nnN` This macro lets #3 to the value of property #2 of article #1. It makes it an empty definition if the property is unset.

```

285 \cs_new:Nn \__yoin_yoinforeach_article_metaitem:nn {
286   \yoin_yoinadd_prop_item:nn { #1 } { article-#2 }
287 }

```

`\__yoin_yoinforeach:nn` `\yoinForEach` itself just sets the keys (in a group to make things local) and then calls `\@@_yoinforeach_article:nn` on each article.

```

288 \cs_new_protected:Nn \yoin_yoinforeach:nn {
289   \group_begin:
290   \keys_set:nn { yoin / yoinforeach } { #1 }
291   \seq_map_inline:Nn \g_yoin_yoinadd_seq { \__yoin_yoinforeach_article:nn { ##1 } { #2 } }
292   \group_end:
293 }

```

`\__yoin_yoinshell_runforarticle:nn` If the tag passed to `onlytag` of `\RunForEach` is identical to the tag of the article or if any of them is not set, we do what should be done, otherwise nothing is done (the tags do not match). We only extract the prop to publically available macros like `\Article`, `\Jobname` etc. (in a group to make this local), and then run the command in shell escape.

```

294 \cs_new_protected:Nn \__yoin_yoinforeach_article:nn {
295   \prop_get:cnN { \yoin_yoinadd_prop:n { #1 } } { tag } \l__yoin_yoinforeach_article_tag_tl
296   \bool_if:nT {
297     \quark_if_no_value_p:N \l__yoin_yoinforeach_article_tag_tl
298     ||
299     \quark_if_no_value_p:N \l__yoin_yoinforeach_onlytag_tl
300     ||
301     \tl_if_eq_p:NN \l__yoin_yoinforeach_onlytag_tl \l__yoin_yoinforeach_article_tag_tl
302   }
303   {
304     \group_begin:
305     \DeclareDocumentCommand \Meta { m } { \__yoin_yoinforeach_article_metaitem:nn { #1 } { ##1 } }
306     \__yoin_yoinforeach_article_keyfromprop:nnN { #1 } { article } \Article
307     \__yoin_yoinforeach_article_keyfromprop:nnN { #1 } { jobname } \Jobname
308     \__yoin_yoinforeach_article_keyfromprop:nnN { #1 } { firstpage } \FirstPage
309     #2

```

```

310     \group_end:
311   }
312 }

```

`\yoinForEach` One optional key–val argument, one mandatory argument — the text to be typeset.

```

313 \NewDocumentCommand \yoinForEach { 0{ } +m } {
314   \yoin_yoinforeach:nn { #1 } { #2 }
315 }

```

## 8 Article setting stuff (undocumented)

Information to be stored in an auxiliary file.

```

316 \cs_new_protected:Nn \__yoin_article_write_keyval:nn {
317   \iow_now:Nn \g__yoin_article_dotyoin_iow { #1 ~ = ~ #2 , }
318 }
319 \cs_generate_variant:Nn \__yoin_article_write_keyval:nn { nx, nV }
320
321 \cs_new_protected:Nn \yoin_article_write_meta:nn {
322   \__yoin_article_write_keyval:nn { article-#1 } { #2 }
323 }
324
325 \cs_new_protected:Nn \yoin_article_writekeys: {
326   \__yoin_article_write_keyval:nV { jobname } \c_job_name_tl
327   \__yoin_article_write_keyval:nx { totpages } { \ztotpages }
328   \__yoin_article_write_keyval:nV { currrdir } \l_yoin_article_currrdir_tl
329   \__yoin_article_write_keyval:nx { firstpage } { \int_use:N \l_yoin_article_firstpage_int }
330 }
331
332 \prop_new:N \l__yoin_article_readkeys_prop
333
334 \cs_new_protected:Nn \yoin_article_set_readkey:nn {
335   \prop_put:Nnn \l__yoin_article_readkeys_prop { #1 } { #2 }
336 }
337 \cs_generate_variant:Nn \yoin_article_set_readkey:nn { Vn }
338
339 \int_new:N \l_yoin_article_firstpage_int
340 \int_set:Nn \l_yoin_article_firstpage_int { 1 }
341
342 \keys_define:nn { yoin / toarticle } {

```

```

343 firstpage .code:n =
344     \int_set:Nn \l_yoin_article_firstpage_int { #1 }
345     \yoin_article_set_readkey:nn { firstpage } { #1 }
346     ,
347
348 parent .code:n =
349     \file_if_exist:nT { ../ #1 .yoin } {
350         \yoin_keys_set_from_file:nn { yoin / toarticle } { ../ #1 .yoin }
351     }
352     \yoin_article_set_readkey:nn { parent } { #1 }
353     ,
354
355 unknown .code:n =
356     \yoin_article_set_readkey:Vn \l_keys_key_tl { #1 }
357     ,
358 }
359
360 \bool_new:N \g__yoin_article_readkeys_bool
361 \bool_gset_true:N \g__yoin_article_readkeys_bool
362
363 \cs_new_protected:Nn \yoin_article_readkeys: {
364     \bool_if:NT \g__yoin_article_readkeys_bool {
365         \file_if_exist:nT { ../ \l_yoin_article_currdir_tl .yoin1 } {
366             \yoin_keys_set_from_file:nn { yoin / toarticle } { ../ \l_yoin_article_currdir_tl .yoin1 }
367         }
368     }
369     \bool_gset_false:N \g__yoin_article_readkeys_bool
370 }
371
372 \cs_new:Nn \yoin_article_meta:n {
373     \prop_item:Nn \l__yoin_article_readkeys_prop { meta-#1 }
374 }
375
376 \cs_new:Nn \yoin_article_meta_gset_tl_default:Nnn {
377     \prop_get:NnNF \l__yoin_article_readkeys_prop { meta-#2 } #1 {
378         \tl_set:Nn #1 { #3 }
379     }
380 }
381
382 \NewDocumentCommand \yoinArticleMeta { m } {

```

```

383   \yoin_article_meta:n { #1 }
384 }
385
386 \tl_new:N \l__yoin_article_tmpa_tl
387 \seq_new:N \l__yoin_article_tmpa_seq
388 \tl_new:N \l_yoin_article_currdir_tl
389 \cs_generate_variant:Nn \regex_extract_once:nnN { nV }
390 \cs_new_protected:Nn \yoin_article_getcurrdir:N {
391   \tl_set:Nx \l__yoin_article_tmpa_tl { \currfileabsdir }
392   \regex_extract_once:nVN { /([~/]+)/\Z } \l__yoin_article_tmpa_tl \l__yoin_article_tmpa_seq
393   \seq_get_right:NN \l__yoin_article_tmpa_seq #1
394 }
395
396 \iow_new:N \g__yoin_article_dotyoin_iow
397 \bool_if:NT \g_yoin_article_bool {
398   \yoin_article_getcurrdir:N \l_yoin_article_currdir_tl
399   \iow_open:Nn \g__yoin_article_dotyoin_iow { \l_yoin_article_currdir_tl .yoin }
400   \yoin_article_readkeys:
401   \AtBeginDocument {
402     \setcounter { page } { \l_yoin_article_firstpage_int }
403     \yoin_article_writekeys:
404   }
405 }

```

## 9 yoinProcess (undocumented)

`msg: forceopenanyright` Error message for an article having both `forceopenany` and `forceopenright` set.

```

406 \msg_new:nnn { yoin } { forceopenanyright }
407   { The ~ article ~ '#1' ~ has ~ both ~ 'forceopenany' ~ and ~ 'forceopenright' ~ keys ~ set. }

```

`\yoin_yoinprocess:n` The key macro of the package, to some sense. It takes care of the page numbering of the articles, proper placement of stuff in twoside environment, etc.

```

408 \cs_new_protected:Nn \yoin_yoinprocess:n {
  Set the appropriate keys (this may be changed later and moved to yoin/general keys.
409   \keys_set:nn { yoin / yoinprocess } { #1 }
  Finish the current page if it's started.
410   \clearpage

```

Go to the right page number. This depends on two parameters, cleardoublepage and setpagenumber, the dependence is explained in each of the 4 cases.

```
411 \bool_if:NTF \l__yoin_yoinprocess_cleardoublepage_bool {
412   \bool_if:NTF \l__yoin_yoinprocess_setpagenumber_bool {
```

Case cleardoublepage, setpagenumber. In this case, an empty page is added as necessary to keep the parity of page numbers. For instance, if setpagenumber=110 and last page number is 4, an empty page is added so that there are no two consecutive even pages. The check is on the parity of the sum of the two numbers. The macro \\_\_yoin\_yoinprocess\_clearonepage: uses the code of L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>'s \cleardoublepage for creating the necessary empty page.

```
413   \int_if_odd:nT { \value { page } + \l__yoin_yoinprocess_setpagenumber_int } {
414     \__yoin_yoinprocess_clearonepage:
415   }
416   \setcounter { page } { \int_use:N \l__yoin_yoinprocess_setpagenumber_int }
417 } {
```

Case cleardoublepage, nosetpagenumber. We simply do a cleardoublepage. Note that \\_\_yoin\_yoinprocess\_cleardoublepage: modifies the value of \g\_@@\_page\_int in a useless way at this place, but we will override the value anyway.

```
418   \__yoin_yoinprocess_cleardoublepage:
419 }
420 }
```

Case nocleardoublepage, setpagenumber. We simply set the page number.

```
421 \bool_if:NTF \l__yoin_yoinprocess_setpagenumber_bool {
422   \setcounter { page } { \int_use:N \l__yoin_yoinprocess_setpagenumber_int }
423 }
```

Case nocleardoublepage, nosetpagenumber. No adjustment is needed in this case.

```
424 \prg_do_nothing:
425 }
426 }
```

Here, the loop through the articles starts. First, set the internal counter for the page number; this is necessary because if the output of the process is suppressed by the key output=false, we need to keep track of the page number manually.

```
427 \int_gset:Nn \g__yoin_page_int { \value { page } }
428 \seq_map_inline:Nn \g_yoin_yoinadd_seq {
```

Handling of even/odd/pages. First, issue an error if both addarticle/forceopenany and addarticle/forceopenright are set.

```
429 \bool_if:nT {
430   \int_compare_p:nNn {
431     \yoin_yoinadd_prop_item:nn { ##1 } { forceopenany }
432     + \yoin_yoinadd_prop_item:nn { ##1 } { forceopenright }
433   } = { 2 }
434 }
```



```

435     \msg_error:nnn { yoin } { forceopenanyright } { ##1 }
436 }

```

Then, we call `cleardoublepage` (our internal variant) if: either `forceopenright` is true; or `openright` is true and `forceopenright` is false.

```

437 \bool_if:NT {
438   \int_compare_p:nNn { \yoin_yoinadd_prop_item:nn { ##1 } { forceopenright } } = { 1 }
439   || (
440     \l__yoin_yoinprocess_openright_bool
441     &&
442     \int_compare_p:nNn { \yoin_yoinadd_prop_item:nn { ##1 } { forceopenany } } = { 0 }
443   )
444 } {
445   \__yoin_yoinprocess_cleardoublepage:
446 }

```

If output is true, we use `\includepdf` to include the PDF of the article.

```

447 \bool_if:NT \l__yoin_yoinprocess_output_bool {
448   \includepdf [ pages = - ] { ##1 / \yoin_yoinadd_prop_item:nn { ##1 } { jobname } .pdf }
449 }

```

Into file `./<articlename>.yoin1` we save the data to be transferred to the article: the first page number (possibly 1 if `alwayspageone` key is set) and the name of the this document.

```

450 \iow_open:Nn \g__yoin_yoinprocess_iow { ##1 .yoin1 }
451 \bool_if:NTF \l__yoin_yoinprocess_alwayspageone_bool {
452   \iow_now:Nx \g__yoin_yoinprocess_iow { firstpage ~ = ~ 1 , }
453 } {
454   \iow_now:Nx \g__yoin_yoinprocess_iow { firstpage ~ = ~ \int_use:N \g__yoin_page_int , }
455 }
456 \iow_now:Nx \g__yoin_yoinprocess_iow { parent ~ = ~ \jobname , }
457 \iow_close:N \g__yoin_yoinprocess_iow

```

Update our internal page counter.

```

458 \int_gadd:Nn \g__yoin_page_int { \yoin_yoinadd_prop_item:nn { ##1 } { totpages } }
459 }
460 }

```

`\yoinProcess` Public wrapper around the  $\text{\LaTeX}$ 3 version.

```

461 \DeclareDocumentCommand \yoinProcess { 0{ } } { \yoin_yoinprocess:n { #1 } }

```

```

462
463 \int_new:N \g__yoin_page_int
464 \iow_new:N \g__yoin_yoinprocess_iow
465
466 \cs_new_protected:Nn \__yoin_yoinprocess_cleardoublepage: {
467   \bool_if:NT \l__yoin_yoinprocess_output_bool { \cleardoublepage }
468   \int_if_even:nT { \g__yoin_page_int } { \int_gincr:N \g__yoin_page_int }
469 }
470
471 \cs_new_protected:Nn \__yoin_yoinprocess_clearonepage: {
472   \bool_if:NT \l__yoin_yoinprocess_output_bool {
473     \hbox {} \newpage \if@twocolumn \hbox {} \newpage \fi
474   }
475   \int_gincr:N \g__yoin_page_int
476 }
477
478 \bool_new:N \l__yoin_yoinprocess_cleardoublepage_bool
479 \bool_new:N \l__yoin_yoinprocess_output_bool
480 \bool_new:N \l__yoin_yoinprocess_openright_bool
481 \bool_new:N \l__yoin_yoinprocess_alwayspageone_bool
482 \bool_new:N \l__yoin_yoinprocess_setpagenumber_bool
483 \int_new:N \l__yoin_yoinprocess_setpagenumber_int
484 \keys_define:nn { yoin / yoinprocess } {
485
486   cleardoublepage .bool_set:N = \l__yoin_yoinprocess_cleardoublepage_bool ,
487   cleardoublepage .initial:n = { false },
488
489   output .bool_set:N = \l__yoin_yoinprocess_output_bool ,
490   output .initial:n = { true },
491
492   openright .bool_set:N = \l__yoin_yoinprocess_openright_bool ,
493   openany .bool_set_inverse:N = \l__yoin_yoinprocess_openright_bool ,
494   openright .initial:n = { false },
495
496   setpagenumber .code:n =
497     \str_if_eq:nnTF { #1 } { false } {
498       \bool_set_false:N \l__yoin_yoinprocess_setpagenumber_bool
499     } {
500       \bool_set_true:N \l__yoin_yoinprocess_setpagenumber_bool
501       \int_set:Nn \l__yoin_yoinprocess_setpagenumber_int { #1 }

```

```

502     }
503     ,
504     setpagenumber .initial:n = { false },
505
506     alwayspageone .bool_set:N = \l__yoin_yoinprocess_alwayspageone_bool ,
507     alwayspageone .initial:n = { false },
508
509 }
510

```

## 10 Experimental

```

\bla
511 \cs_new:Nn \yoin_blabla: {
512     Blabla
513 }
514
515 </package>

```