

The `latex-lab-table` package

Changes related to the tagging of tables

Frank & Ulrike, L^AT_EX Project*

v0.85d 2023-10-30

Abstract

The following code implements a first draft for the tagging of tables. It still has a large number of limitations and restrictions!

Contents

1	Documentation	2
2	Limitations	3
3	Introduction	4
4	Technical details and problems	4
	4.1 TODOs	4
5	Implementation	4
	5.1 Variables	5
	5.2 Sockets	5
	5.3 Environments	11
	5.4 Interfaces to tagging	12
	5.4.1 Tagging helper commands	12
	5.4.2 Disabling/enabling	12
	5.4.3 Header support	13
	5.5 Changes to array commands	15
	5.6 <code>longtable</code>	23
	5.7 <code>tabularx</code>	29

*Initial implementation done by Frank Mittelbach

1 Documentation

In \LaTeX the word `table` is used as the name of the float environment that can contain a data table¹ along with a caption and some additional text. The environments for actual data tables have various names like `tabular`, `tabular*`, `tabularx` and `longtable`—the last should not be used inside a float and supports its own caption command.

In this documentation “table” always means such data tables and not the float environment.

Tagging of tables is on one side doesn’t look very difficult: one only has to surround rows and cells by TR and TH or TD structures. But there are difficulties:

- One is that over the years various packages related to tables have been written that all change some of the internals. Inserting the tagging commands and testing all the variants and various nestings is not trivial.
- The other difficulty is that most of the existing environments to create tables do not know the concept of headers as a semantic structures.
- Headers are only produced through visual formatting, e.g., by making them bold or by underlying some color. But accessible tables need headers (and the PDF/UA standards requires them) and this means that additional syntax to declare headers (when they can’t be guessed) must be developed. This is still an area for research.

Right now, this module therefore does implement only some basic support for the tagging of tables. A list of the known limitations is shown below.

The module is not loaded automatically (i.e., not yet integrated into any `phase-XX`) and by itself it doesn’t activate tagging. For experimenting with table tagging it is therefore best to load it in combination with `phase-III` in `\DocumentMetadata`, i.e.:

```
\DocumentMetadata{testphase={phase-III,table}}
```

It will then automatically tag all table environments it already supports with the exception of tables in the header and footer of the page (where tagging is disabled). Such tables can be nested.

If a table should not be tagged as table, for example because it is merely used to produce a layout or because it is a not yet (fully) supported table structure, the tagging can be disabled with `\tagpdfsetup{table-tagging=false}`.

Inside cells the automatic tagging of paragraphs is disabled with the exception of p/m/b-type cells.

Rows do not need to contain a full number of `&`, missing cells are automatically added with an empty TD-structure.

There is some basic support² for headers. With

```
\tagpdfsetup{table-header-rows={\list of row numbers}}
```

you can declare which (absolute) row numbers should be tagged as header rows. It applies to all tables until it is changed to a different list of row numbers or undone by setting the key to `\empty`. A row number can be negative, then the counting starts from the end of the table. There is no support for header columns yet. In a `longtable` the code will

¹But it does not really have to, you can put other material into such environments.

²This is not meant to be the final interface, though.

currently use the `\endhead` or `\endfirsthead` rows as header if one of these commands has been used and in that case the code ignores a `table-header-rows` setting.

You should not insert meaningful text with `!\{...\}` or `@{...\}` or `\noalign` between the rows or columns of the table. With `pdflatex` such text will be unmarked, with `lualatex` it will be marked as artifact. Either case means that it will be currently ignored in the structure.³

As mentioned below the `colortbl` doesn't yet work properly with the tagging, but once it does, then colors inside the table will probably be simply ignored (at least initially). If such a color has a semantic meaning (like "important value") this meaning will be lost.

Feedback and problems with the code can be reported at <https://github.com/latex3/tagging-project> either in form of explicit issues or as a "discussion topic", whatever works best.

2 Limitations

- The code loads the `array` package and so does not work without it (that is not really a limitation, but can affect existing tables).
- It supports only a restricted number of tables types. Currently `tabular`, `tabular*`, `tabularx`, and `longtable`.
- the `array` environment is assumed to be part of math and tagging as a table is disabled for it.
- Row spans are not yet supported (and the `multirow` package is untested).
- The `colortbl` package breaks tagging if there are nested tables. It also breaks the filling up of incomplete rows.
- The `tabularray` package use a completed different method to create tables and will not be supported by this code.
- The `nicematrix` package is currently incompatible.
- Most other packages related to tables in \LaTeX are not yet tested, that includes packages that change rules like `booktabs`, `hhline`, `arydshln`, `hvdashln`.
- `longtable` currently only works with `lualatex`. With other engines it breaks as its output routine clashes with the code which closes open MC-chunks at pagebreaks and if this is resolved there will probably be problems with the head and foot boxes (but this can't be tested currently).
- Not every table should be tagged as a Table structure, often they are only used as layout help, e.g. to align authors in a title pages. In such uses the tagging of the table must be deactivated with `\tagpdfsetup{table-tagging=false}`.
- Only simple header rows are currently supported. Columns and complex headers with subheaders will be handled later as that needs some syntax changes. Tables with more than one header row are probably not pdf/UA as the headers array in the cells is missing.

³While it is theoretically possible to collect such text and move it into a structure it would require manual markup from the author to clarify where this text belongs too.

- A `longtable \caption` is currently simply formatted as a multicolumn and not tagged as a `Caption` structure.
- The `caption` package will quite probably break the `longtable` caption.
- The setup for `longtable` requires lots of patches to internal `longtable` commands and so can easily break if other packages try to patch `longtable` too.
- The `longtable` environment supports footnotes in p-type columns, but it hasn't been tested yet if this works also with the tagging code.
- The code is quite noisy and fills the log with lots of messages.⁴

3 Introduction

4 Technical details and problems

The implementation has to take care of various details.

4.1 TODOs

- Test `array-006-longtable.lvt` and `array-007-longtable.lvt` have errors with `pdftex` (para tagging)
- Instead of before/after hooks we should add sockets directly into the code.
- Debugging code and messages must be improved.
- Cells need an `Headers` array.
- Row spans should be supported (but perhaps need syntax support)
- Longtable captions should be properly supported.
- Handle p-cells better. para-tagging should probably be enabled, but `Part` can't be a child of `TD`, so this should probably be changed to `Div` here. Also there is a stray `MC` somewhere.
- More packages must be tested.

5 Implementation

```

1 <@@=tbl>
2 <*package>
3 \ProvidesExplPackage {latex-lab-testphase-table} {\ltblabtbldate} {\ltblabtblversion}
4 {Code related to the tagging of tables}

```

This builds on `array` so we load it by default:

```

5 \RequirePackage{array}

```

⁴Helpful for us at this stage.

5.1 Variables

```

\l__tbl_celltag_tl      This is for the celltag, e.g. TD or TH:
\l__tbl_rowtag_tl      6 \tl_new:N \l__tbl_celltag_tl
\l__tbl_cellattribute_tl 7 \tl_set:Nn \l__tbl_celltag_tl {TD}
\l__tbl_rowattribute_tl  For the rowtag, probably always TR:
\g__tbl_missingcells_int 8 \tl_new:N \l__tbl_rowtag_tl
\l__tbl_tmpa_clist      9 \tl_set:Nn \l__tbl_rowtag_tl {TR}
\l__tbl_tmpa_seq
\l__tbl_tmpa_tl

```

And here cell and row attributes:

```

10 \tl_new:N \l__tbl_cellattribute_tl
11 \tl_set:Nn \l__tbl_cellattribute_tl {}
12 \tl_new:N \l__tbl_rowattribute_tl
13 \tl_set:Nn \l__tbl_rowattribute_tl {}

```

This will contain the number of missing cells used:

```

14 \int_new:N \g__tbl_missing_cells_int

```

Temp variables

```

15 \clist_new:N \l__tbl_tmpa_clist
16 \seq_new:N \l__tbl_tmpa_seq
17 \tl_new:N \l__tbl_tmpa_tl

```

(End of definition for `\l__tbl_celltag_tl` and others.)

5.2 Sockets

The code uses a number of sockets to inject the tagging commands. These can be easily set to a noop-plug in case the automated tagging is not wanted. At first sockets for the begin and end of cells and rows

```

tagsupport/tblcell/begin (socket)
tagsupport/tblcell/end (socket) 18 \NewSocket{tagsupport/tblcell/begin}{0}
tagsupport/tblrow/begin (socket) 19 \NewSocket{tagsupport/tblcell/end}{0}
tagsupport/tblrow/end (socket) 20 \NewSocket{tagsupport/tblrow/begin}{0}
21 \NewSocket{tagsupport/tblrow/end}{0}

```

`tagsupport/tbl/init (socket)` This socket should be at the begin of the table, inside a group. It is meant for settings like disabling paratagging. This socket can perhaps be merged later into the begin-sockets when they are no longer added as hooks but in the environment definitions.

```

22 \NewSocket{tagsupport/tbl/init}{0}

```

`tagsupport/tbl/finalize (socket)` To fine tune the structure (change cells to header cells, remove unwanted structures, move a foot to the end, etc.) we also need a socket that is executed at the end of the table but *before* all the variables are restored to the outer or default values. The code in the socket can make assignments, but probably shouldn't do typesetting and not write whatsits.

```

23 \NewSocket{tagsupport/tbl/finalize}{0}

```

`tagsupport/tbl/finalize/longtable (socket)` `longtable` needs its own socket to fine tune the structure. Simply switching the plug in the previous socket interferes with enabling/disabling the tagging.

```

24 \NewSocket{tagsupport/tbl/finalize/longtable}{0}

```

`\tagssupport/tblhmode/begin (socket)` These sockets are used in the begin and end code of environments, to allow a fast enabling and disabling of the tagging. We distinguish between tables that can be used inside paragraphs and standalone tables like `longtable`.

```

25 \NewSocket{tagssupport/tblhmode/begin}{0}
26 \NewSocket{tagssupport/tblhmode/end}{0}
27 \NewSocket{tagssupport/tblvmode/begin}{0}
28 \NewSocket{tagssupport/tblvmode/end}{0}

```

This are the standard plugs for tagging of cells and rows.

TD (*plug*)

```

29 \NewSocketPlug{tagssupport/tblcell/begin}{TD}
30 {
31   \tag_struct_begin:n
32   {
33     tag           =\l__tbl_celltag_tl,
34     attribute-class =\l__tbl_cellattribute_tl
35   }
36   \seq_gput_right:Ne \g__tbl_struct_cur_seq { \tag_get:n {struct_num} }

```

we store the cells of multicolumns as negative number. This allow to skip them or to use them as needed.

```

37   \int_step_inline:nn { \g__tbl_span_tl - 1 }
38   {
39     \seq_gput_right:Ne \g__tbl_struct_cur_seq { -\tag_get:n {struct_num} }
40   }
41   \tag_mc_begin:n{}
42 }

```

TD (*plug*)

```

43 \NewSocketPlug{tagssupport/tblcell/end}{TD}
44 {
45   \tag_mc_end:
46   \tag_struct_end:
47 }

```

In p-columns we need a slightly different plug which reactivates the paragraph tagging. tagging

TD (*plug*)

```

48 \NewSocketPlug{tagssupport/tblcell/begin}{TDpbox}
49 {
50   \tag_struct_begin:n
51   {
52     tag           =\l__tbl_celltag_tl,
53     attribute-class =\l__tbl_cellattribute_tl
54   }
55   \seq_gput_right:Ne \g__tbl_struct_cur_seq { \tag_get:n {struct_num} }
56   \int_step_inline:nn { \g__tbl_span_tl - 1 }
57   {
58     \seq_gput_right:Ne \g__tbl_struct_cur_seq { -\tag_get:n {struct_num} }
59   }
60   \tagpdfparaOn
61   \tl_set:Nn \l__tag_para_main_tag_tl {Div}
62 }

```

TD (*plug*)

```
63 \NewSocketPlug{tag-support/tblcell/end}{TDpbox}
64 {
65   \tag_struct_end:
66 }
```

TR (*plug*)

```
67 \NewSocketPlug{tag-support/tblrow/begin}{TR}
68 {
69   \seq_gclear:N \g__tbl_struct_cur_seq
70   \tag_struct_begin:n
71   {
72     tag           =\l__tbl_rowtag_tl,
73     attribute-class=\l__tbl_rowattribute_tl
74   }
75   \seq_gput_right:Ne \g__tbl_struct_rows_seq { \tag_get:n {struct_num} }
76 }
```

TR (*plug*)

```
77 \NewSocketPlug{tag-support/tblrow/end}{TR}
78 {
79   \__tag_tbl_add_missing_cells:n { \g__tbl_missing_cells_int }
80   \seq_gput_right:Ne \g__tbl_struct_cells_seq
81   {
82     \seq_use:Nn \g__tbl_struct_cur_seq {,}
83   }
84   \int_compare:nNnTF { \g__tbl_row_int } = { \seq_count:N\g__tbl_struct_cells_seq }
85   {
86     \typeout
87     {==>~
88       structure-stored-for-row~\int_use:N\g__tbl_row_int :-
89       \seq_use:Nn \g__tbl_struct_cur_seq {,}
90     }
91   }
92   { \ERROR } % should not happen ...
93   \tag_struct_end:
94 }
```

And the plugs for the table as whole. The code can be different for normal tables which can also be used inline and nested and “vmode” tables like longtable.

Table (*plug*) Inside a table we currently only disable paratagging. We assume that these sockets are in an environment group, so there is no need to reenabale paratagging.

```
95 \NewSocketPlug{tag-support/tbl/init}{Table}
96 {
97   \tag_if_active:T
98   {
99     \bool_set_false:N \l__tag_para_bool
100   }
101 }
```

Table (*plug*) This plug will fine tune the structure.

```
102 \NewSocketPlug{tag-support/tbl/finalize}{Table}
```

```

103 {
104   \tbl_set_header_rows:
105 }

```

Table (plug) This plug will fine tune the structure of longtable.

```

106 \NewSocketPlug{tagssupport/tbl/finalize/longtable}{Table}
107 {

```

If neither `\endhead` nor `\endfirsthead` has been used we use the standard header command:

```

108   \bool_lazy_and:nnTF
109     { \seq_if_empty_p:N \g__tbl_LT@head_rows_seq }
110     { \seq_if_empty_p:N \g__tbl_LT@firsthead_rows_seq }
111     { \tbl_set_header_rows: }

```

Otherwise, if `firsthead` has not been used we use `head`. For this we simple retrieve the row numbers and then call the header command.

```

112   {
113     \seq_if_empty:NNTF \g__tbl_LT@firsthead_rows_seq
114     {
115       \clist_set:Nc \l__tbl_header_rows_clist
116         { \seq_use:Nn \g__tbl_LT@head_rows_seq {,} }
117       \tbl_set_header_rows:
118     }

```

In the other case we use `firsthead`.

```

119   {
120     \clist_set:Nc \l__tbl_header_rows_clist
121       { \seq_use:Nn \g__tbl_LT@firsthead_rows_seq {,} }
122     \tbl_set_header_rows:

```

Additionally we have to remove the head to avoid duplication. The one option here is to remove the rows from the kid sequence of the table (which will lead to orphaned structure elements), the other to make them artifact. For now we use the first option for pdf 1.7 and the second for pdf 2.0.

```

123     \pdf_version_compare:NnTF < {2.0}
124     {
125       \seq_map_inline:Nn \g__tbl_LT@head_rows_seq
126       {
127         \seq_gset_item:cnn
128           {g__tag_struct_kids_ \g__tbl_struct_table_t1 _seq}
129           { ##1 }
130       }

```

Not sure if needed, but if needed we can remove also the P tag. This is currently disabled as it produce warnings. TODO: This needs also a `tagpdf` command which takes care of debug code.

```

131     \t1_set:Nc \l__tbl_tmpa_t1
132       { \seq_item:Nn \g__tbl_struct_rows_seq {##1} }
133     \prop_if_exist:cT
134       { g__tag_struct_ \l__tbl_tmpa_t1 _prop }
135     {
136       %\prop_gremove:cn {g__tag_struct_ \l__tbl_tmpa_t1 _prop} {P}
137     }
138   }

```



```

139     }
140     {
141         \seq_map_inline:Nn \g__tbl_LT@head_rows_seq
142         {
143             \tl_set:Ne \l__tbl_tmpa_tl
144             { \seq_item:Nn\g__tbl_struct_rows_seq {##1} }
145             \prop_if_exist:cT
146             { g__tag_struct_ \l__tbl_tmpa_tl _prop }
147             {
148                 \__tbl_struct_prop_gput:Vnn \l__tbl_tmpa_tl {S}{/Artifact}
149             }
150         }
151     }
152 }
153 }

```

The foot is handled similar, the difference is that we have to move it to the end one of them is not empty, but do nothing if they aren't there.

```

154     \bool_lazy_and:nnF
155     { \seq_if_empty_p:N \g__tbl_LT@foot_rows_seq }
156     { \seq_if_empty_p:N \g__tbl_LT@lastfoot_rows_seq }
157     {

```

If lastfoot is empty move foot to the end.

```

158         \seq_if_empty:NTF \g__tbl_LT@lastfoot_rows_seq
159         {
160             \seq_map_inline:Nn \g__tbl_LT@foot_rows_seq
161             {
162                 \tl_set:Ne \l__tbl_tmpa_tl
163                 {
164                     \seq_item:cn
165                     {g__tag_struct_kids_ \g__tbl_struct_table_tl _seq}
166                     {##1}
167                 }
168                 \seq_gset_item:cnm
169                 {g__tag_struct_kids_ \g__tbl_struct_table_tl _seq}
170                 { ##1 }
171                 {}
172                 \seq_gput_right:cV
173                 {g__tag_struct_kids_ \g__tbl_struct_table_tl _seq}
174                 \l__tbl_tmpa_tl
175             }
176         }

```

If lastfoot is not empty we move that.

```

177     {
178         \seq_map_inline:Nn \g__tbl_LT@lastfoot_rows_seq
179         {
180             \tl_set:Ne \l__tbl_tmpa_tl
181             {
182                 \seq_item:cn
183                 {g__tag_struct_kids_ \g__tbl_struct_table_tl _seq}
184                 {##1}
185             }
186             \seq_gset_item:cnm

```

```

187         {g__tag_struct_kids_ \g__tbl_struct_table_tl _seq}
188         { ##1 }
189         {}
190     \seq_gput_right:cV
191     {g__tag_struct_kids_ \g__tbl_struct_table_tl _seq}
192     \l__tbl_tmpa_tl
193 }

```

and we hide foot

```

194 \pdf_version_compare:NnTF < {2.0}
195 {
196     \seq_map_inline:Nn \g__tbl_LT@foot_rows_seq
197     {
198         \seq_gset_item:cnn
199         {g__tag_struct_kids_ \g__tbl_struct_table_tl _seq}
200         { ##1 }
201         {}

```

Not sure if needed, but if needed we can remove also the P tag. This is currently disabled as it produce warnings. TODO: This needs also a tagpdf command which takes care of debug code.

```

202         \tl_set:Ne \l__tbl_tmpa_tl
203         { \seq_item:Nn\g__tbl_struct_rows_seq {##1} }
204     \prop_if_exist:cT
205     { g__tag_struct_ \l__tbl_tmpa_tl _prop }
206     {
207         %\prop_gremove:cn {g__tag_struct_ \l__tbl_tmpa_tl _prop} {P}
208     }
209 }
210 }
211 {
212     \seq_map_inline:Nn \g__tbl_LT@foot_rows_seq
213     {
214         \tl_set:Ne \l__tbl_tmpa_tl
215         { \seq_item:Nn\g__tbl_struct_rows_seq {##1} }
216     \prop_if_exist:cT
217     { g__tag_struct_ \l__tbl_tmpa_tl _prop }
218     {
219         \__tbl_struct_prop_gput:Vnn \l__tbl_tmpa_tl {S}{/Artifact}
220     }
221     }
222 }
223 }
224 }
225 }

```

Table (*plug*)

```

226 \NewSocketPlug{tagssupport/tblhmode/begin}{Table}
227 {
228     \mode_leave_vertical:
229     \tag_mc_end_push:

```

Close the P-chunk. This assumes that para-tagging is active. For nested tables that is not necessarily true, so we test for it.

```

230     \bool_lazy_and:nnT

```

```

231     { \bool_if_exist_p:N \l__tag_para_bool } { \l__tag_para_bool }
232     { \tag_struct_end:n { text } }
233     \tag_struct_begin:n {tag=Table}
234     \tl_gset:Ne \g__tbl_struct_table_tl { \tag_get:n {struct_num} }
235 }

```

Table (*plug*)

```

236 \NewSocketPlug{tagsupport/tblhmode/end}{Table}
237 {
238     \tag_struct_end:

```

reopen the P-chunk. This assumes that para-tagging is active. For nested tables that is not necessarily true, so we test for it.

```

239     \bool_lazy_and:nnT
240     { \bool_if_exist_p:N \l__tag_para_bool } { \l__tag_para_bool }
241     { \tag_struct_begin:n { tag=\l__tag_para_tag_tl } }
242     \tag_mc_begin_pop:n{}
243 }

```

Table (*plug*)

```

244 \NewSocketPlug{tagsupport/tblvmode/begin}{Table}
245 {
246     \tag_struct_begin:n {tag=Table}
247     \tl_gset:Ne \g__tbl_struct_table_tl { \tag_get:n {struct_num} }
248 }

```

Table (*plug*)

```

249 \NewSocketPlug{tagsupport/tblvmode/end}{Table}
250 {
251     \tag_struct_end:
252     \par
253 }

```

5.3 Environments

Currently only tabular, tabular*, tabularx and longtable. We must use the `before` and `after` hooks as the `end` hook is executed before the end of the last row and then MC are messed up. This means that this sockets should only contain code that doesn't needs to be grouped!

```

254 \AddToHook{env/tabular/before} {\UseSocket{tagsupport/tblhmode/begin}}
255 \AddToHook{env/tabular/after} {\UseSocket{tagsupport/tblhmode/end}}
256 \AddToHook{env/tabular*/before} {\UseSocket{tagsupport/tblhmode/begin}}
257 \AddToHook{env/tabular*/after} {\UseSocket{tagsupport/tblhmode/end}}
258 \AddToHook{env/tabularx/before} {\UseSocket{tagsupport/tblhmode/begin}}
259 \AddToHook{env/tabularx/after} {\UseSocket{tagsupport/tblhmode/end}}
260 \AddToHook{env/longtable/before}{\UseSocket{tagsupport/tblvmode/begin}}
261 \AddToHook{env/longtable/after} {\UseSocket{tagsupport/tblvmode/end}}

```

The array environment is math. So we disable table tagging for now.

```

262 \AddToHook{env/array/begin} {\__tag_tbl_disable:}

```

5.4 Interfaces to tagging

5.4.1 Tagging helper commands

`_tbl_set_colspan:n` This commands takes a number, checks if is larger than one, checks if the colspan attribute exists (we can't predefine an arbitrary number), and updates `\l_tbl_cellattribute_tl`.

```
263 \tag_if_active:T
264 { \cs_generate_variant:Nn \_tag_attr_new_entry:nn {ee} }
265 \cs_new_protected:Npn \_tbl_set_colspan:n #1
266 {
267   \tag_if_active:T
268   {
269     \int_compare:nNnT {#1}>{1}
270     {
271       \prop_get:NeNF \g\_tag_attr_entries_prop
272         {colspan-\int_eval:n{#1}}
273       \l\_tbl_tmpa_tl
274       {
275         \_tag_attr_new_entry:ee
276           {colspan-\int_eval:n{#1}}
277           {/O /Table /ColSpan~\int_eval:n{#1}}
278       }
279       \tl_set:Ne \l\_tbl_cellattribute_tl
280         {colspan-\int_eval:n{#1}}
281     }
282   }
283 }
```

(End of definition for _tbl_set_colspan:n.)

5.4.2 Disabling/enabling

For now we have only the option true/false but this will probably be extended to allow different setups like first row header etc.

`_tag_tbl_disable:`

```
284 \cs_new_protected:Npn \_tag_tbl_disable:
285 {
286   \AssignSocketPlug{tag-support/tblcell/begin}{noop}
287   \AssignSocketPlug{tag-support/tblcell/end}{noop}
288   \AssignSocketPlug{tag-support/tblrow/begin}{noop}
289   \AssignSocketPlug{tag-support/tblrow/end}{noop}
290   \AssignSocketPlug{tag-support/tbl/init}{noop}
291   \AssignSocketPlug{tag-support/tbl/finalize}{noop}
292   \AssignSocketPlug{tag-support/tbl/finalize/longtable}{noop}
293   \AssignSocketPlug{tag-support/tblhmode/begin}{noop}
294   \AssignSocketPlug{tag-support/tblhmode/end}{noop}
295   \AssignSocketPlug{tag-support/tblvmode/begin}{noop}
296   \AssignSocketPlug{tag-support/tblvmode/end}{noop}
297 }
```

(End of definition for _tag_tbl_disable:.)

_tag_tbl_enable:

```
298 \\cs_new_protected:Npn \\_tag_tbl_enable:
299 {
300   \\AssignSocketPlug{tagsupport/tblcell/begin}{TD}
301   \\AssignSocketPlug{tagsupport/tblcell/end}{TD}
302   \\AssignSocketPlug{tagsupport/tblrow/begin}{TR}
303   \\AssignSocketPlug{tagsupport/tblrow/end}{TR}
304   \\AssignSocketPlug{tagsupport/tbl/init}{Table}
305   \\AssignSocketPlug{tagsupport/tbl/finalize}{Table}
306   \\AssignSocketPlug{tagsupport/tbl/finalize/longtable}{Table}
307   \\AssignSocketPlug{tagsupport/tblhmode/begin}{Table}
308   \\AssignSocketPlug{tagsupport/tblhmode/end}{Table}
309   \\AssignSocketPlug{tagsupport/tblvmode/begin}{Table}
310   \\AssignSocketPlug{tagsupport/tblvmode/end}{Table}
311 }
```

(End of definition for _tag_tbl_enable:.)

```
312 % TODO decide about key name
313 \\keys_define:nn { \\_tag / setup }
314 {
315   table-tagging .choices:nn = { true, on }
316   { \\_tag_tbl_enable: },
317   table-tagging .choices:nn = { false, off }
318   { \\_tag_tbl_disable: },
319   table-tagging .default:n = true,
320   table-tagging .initial:n = true
321 }
```

Table tagging should be disabled in the head and foot.

```
322 \\AddToHook{begindocument}
323 {
324   \\cs_if_exist:NT \\@kernel@before@head
325   {
326     \\tl_put_right:Nn \\@kernel@before@head {\\_tag_tbl_disable:}
327     \\tl_put_right:Nn \\@kernel@before@foot {\\_tag_tbl_disable:}
328   }
329 }
```

5.4.3 Header support

Accessible table must have header cells declaring the meaning of the data in a row or column. To allow a data cell to find it header cell(s) a number of things must be done:

- every cell meant as a header should use the tag TH.
- header cells should have a `Scope` attribute with the value `Column`, `Row` or `Both`. This is not needed in the first row or column of a table.
- For more complex cases both TD and TH cell can contain a `Headers` attribute, that is an array of IDs of TH cell.

For now we support only header rows.

At first we define attributes for the three standard cases: We delay to begin document as we can't know if tagpdf is already loaded.

```

330 \AddToHook{begindocument}
331 {
332   \tag_if_active:T
333   {
334     \tagpdfsetup
335     {
336       newattribute =
337         {TH-col}{/O /Table /Scope /Column},
338       newattribute =
339         {TH-row}{/O /Table /Scope /Row},
340       newattribute =
341         {TH-both}{/O /Table /Scope /Both},
342     }

```

And we put all three into the class map (perhaps the next tagpdf should do that directly with newattribute):

```

343     \seq_gput_left:Ne\g__tag_attr_class_used_seq
344     {\pdf_name_from_unicode_e:n{TH-col}}
345     \seq_gput_left:Ne\g__tag_attr_class_used_seq
346     {\pdf_name_from_unicode_e:n{TH-row}}
347     \seq_gput_left:Ne\g__tag_attr_class_used_seq
348     {\pdf_name_from_unicode_e:n{TH-both}}
349   }
350 }
351

```

\l__tbl_header_rows_clist This holds the numbers of the header rows. Negative numbers are possible and count from the back.

```

352 \clist_new:N \l__tbl_header_rows_clist

```

__tbl_set_header_rows: TEMP: Next tagpdf will have the right command which also updates the debug info. For now a temporary command:

```

353 \cs_if_free:NTF \__tag_struct_prop_gput:nnn
354 {
355   \cs_new_protected:Npn \__tbl_struct_prop_gput:nnn #1#2#3
356   { \prop_gput:cnn { g__tag_struct_#1_prop }{#2}{#3} }
357 }
358 { \cs_new_protected:Npn \__tbl_struct_prop_gput:nnn #1#2#3
359   { \__tag_struct_prop_gput:nnn {#1}{#2}{#3} }
360 }
361 \cs_generate_variant:Nn \__tbl_struct_prop_gput:nnn {nne,Vnn}
362 \cs_new_protected:Npn \__tbl_set_header_rows:
363 {
364   \clist_map_inline:Nn \l__tbl_header_rows_clist
365   {
366     \clist_set:Ne\l__tbl_tmpa_clist
367     {
368       \seq_item:Nn \g__tbl_struct_cells_seq {##1}
369     }
370     \clist_map_inline:Nn \l__tbl_tmpa_clist
371     {

```

We can have negative numbers in the list from the multicolumn.

```

372         \prop_if_exist:cT { g__tag_struct_####1_prop }
373         {
374         \__tbl_struct_prop_gput:nnn{ ####1 }{S}{/TH}

```

This need refinement once row headers (and perhaps other attributes) are used too, but for now it should be ok.

```

375         \prop_get:cnNTF
376         { g__tag_struct_####1_prop }
377         { C }
378         \l__tbl_tmpa_tl
379         {\__tbl_struct_prop_gput:nne{ ####1 }{C}{[/TH-col-\l__tbl_tmpa_tl]} }
380         {\__tbl_struct_prop_gput:nnn{ ####1 }{C}{/TH-col}}
381     }
382 }
383 }
384 }

```

(End of definition for `__tbl_set_header_rows:`)

And some key support:

```

385 % TODO decide about key name
386 \keys_define:nn { __tag / setup }
387 {
388     table-header-rows .clist_set:N = \l__tbl_header_rows_clist
389 }

```

5.5 Changes to array commands

`__tbl_show_curr_cell_data:` Show the row/column index and span count for current table cell for debugging.

```

390 \cs_new_protected:Npn \__tbl_show_curr_cell_data: {
391     \typeout { ==>~ current~cell~data:~
392             \int_use:N \g__tbl_row_int ,
393             \int_use:N \g__tbl_col_int ,
394             \g__tbl_span_tl
395     }
396 }

```

(End of definition for `__tbl_show_curr_cell_data:`)

`\insert@column` `\insert@column` is defined in array, here only the two sockets are inserted.

```

397 \def\insert@column{%
398     \__tbl_show_curr_cell_data:
399     \UseSocket{tag-support/tblcell/begin}%
400     \the@toks \the \@tempcnta
401     \ignorespaces \@sharp \unskip
402     \the@toks \the \count@ \relax
403     \UseSocket{tag-support/tblcell/end}%
404 }

```

(End of definition for `\insert@column`. This function is documented on page ??.)

`\@classz`

```
405 \def\@classz{\@classx
406   \@tempcnta \count@
407   \prepnext@tok
408   \@addtopreamble{\ifcase \@chnum
409     \hfil
410     \hskip1sp%
411     \d@llarbegin
412     \insert@column
413     \d@llarend \do@row@strut \hfil \or
414     \hskip1sp\d@llarbegin \insert@column \d@llarend \do@row@strut \hfil \or
415     \hfil\hskip1sp\d@llarbegin \insert@column \d@llarend \do@row@strut \or
416     \setbox\ar@mcellbox\vbox
417     \@startpbox{\@nextchar}
418     \AssignSocketPlug{tagsupport/tblcell/begin}{TDpbox}
419     \AssignSocketPlug{tagsupport/tblcell/end}{TDpbox}
420     \insert@column \@endpbox
421     \ar@align@mcell
422     \do@row@strut \or
423     \vtop \@startpbox{\@nextchar}
424     \AssignSocketPlug{tagsupport/tblcell/begin}{TDpbox}
425     \AssignSocketPlug{tagsupport/tblcell/end}{TDpbox}
426     \insert@column \@endpbox\do@row@strut \or
427     \vbox \@startpbox{\@nextchar}
428     \AssignSocketPlug{tagsupport/tblcell/begin}{TDpbox}
429     \AssignSocketPlug{tagsupport/tblcell/end}{TDpbox}
430     \insert@column \@endpbox\do@row@strut
431   \fi}\prepnext@tok}
```

(End of definition for \@classz. This function is documented on page ??.)

`\@array` We modificate the `\@array` from array.

```
\@@array 432 \def\@array[#1]#2{%
433   \@tempdima \ht \strutbox
434   \advance \@tempdima by\extrarowheight
435   \setbox \@arstrutbox \hbox{\vrule
436     \@height \arraystretch \@tempdima
437     \@depth \arraystretch \dp \strutbox
438     \@width \z@}%
```

The total number of table columns of the current table is determined in `__tbl_determine_table_cols:` but this is called in a group, so local settings do not survive. Thus, to save away the outer value of `\g__tbl_table_cols_tl` we do it before the group.

```
439   \tl_set_eq:NN \l__tbl_saved_table_cols_tl \g__tbl_table_cols_tl
440   \begingroup
441   \@mkpream{#2}
```

Next call has to happen immediately after `\@mkpream` because it uses implementation details from that.

```
442   \__tbl_determine_table_cols:
443   \xdef\@preamble{%
444     \noexpand
```


`\ialign` in the original definition is replaced by `\ar@ialign` defined below.

```
445 \ar@ialign
446 \@halignto
447 \bgroup \@arstrut
```

A socket is inserted

```
448 \UseSocket{tagssupport/tblrow/begin}%
```

At the start of the preamble for the first column we set `\g__tbl_col_int` to 1 as we are no longer at but in the first column. This is done in `__tbl_init_cell_data:`. In later columns this data is updated via `__tbl_update_cell_data:`.

```
449 \__tbl_init_cell_data:
450 \@preamble
451 \tabskip \z@ \cr}%
452 \endgroup
453 \@arrayleft
```

Another socket for tagging. TODO: what about `\arrayleft`?

```
454 \UseSocket{tagssupport/tbl/init}
455 \if #1t\vtop \else \if#1b\ vbox \else \vcenter \fi \fi
456 \bgroup
457 \let \@sharp ##\let \protect \relax
458 \lineskip \z@
459 \baselineskip \z@
460 \m@th
461 \let\\\@arraycr \let\tabularnewline\\\let\par\@empty
462 %\show\@preamble
463 \@preamble}
```

Finally, also set `\@@array` to the new definition:

```
464 \let\@@array\@array
```

(End of definition for `\@array` and `\@@array`. These functions are documented on page ??.)

`__tbl_init_cell_data:`

```
465 \cs_new_protected:Npn \__tbl_init_cell_data: {
466 \int_gset:Nn \g__tbl_col_int {1}
467 \tl_gset:Nn \g__tbl_span_tl {1}
468 }
```

(End of definition for `__tbl_init_cell_data:`.)

`__tbl_update_cell_data:` Updating cell data in columns after the first means we have to increment the `\g__tbl_col_int` by the span count of the previous cell (in case it was a `\multicolumn` and then reset the `\g__tbl_span_tl` to one (as the default).

```
469 \cs_new_protected:Npn \__tbl_update_cell_data: {
470 \int_gadd:Nn \g__tbl_col_int { \g__tbl_span_tl }
471 \tl_gset:Nn \g__tbl_span_tl {1}
472 }
```

(End of definition for `__tbl_update_cell_data:`.)

`_tbl_determine_table_cols:` Current implementation of `\@mkpream` uses the scratch counter `\count@` to keep track of the number of toks registers it needs (2 per column), but this can't be used as it counts also insertings made with `!{}` and `@{}`. So similar as does `longtable` for `\LT@cols` we count the numbers of ambersands instead.

```

473 \cs_new:Npn \_tbl_determine_table_cols: {
474   \seq_set_split:NnV\l__tbl_tmpa_seq {&}\@preamble
475   \tl_gset:Nc \g__tbl_table_cols_tl { \seq_count:N \l__tbl_tmpa_seq }
476   \typeout{==>~ Table~ has~ \g__tbl_table_cols_tl \space columns }
477 }

```

(End of definition for `_tbl_determine_table_cols:.`)

`\@arraycr` Add code that figures out if the current table row is incomplete (not enough `&s`). It can then do extra actions, such as inserting missing cell tags.

```

478 \protected\def\@arraycr{
479   \relax
480   \_tbl_store_missing_cells:n{\@arraycr}
481   %
482   \iffalse{\fi\ifnum 0='}\fi
483   \@ifstar \@xarraycr \@arraycr}

```

(End of definition for `\@arraycr`. This function is documented on page ??.)

`_tbl_store_missing_cells:n`
`_tag_tbl_add_missing_cells:n` The storing and use of the number of missing cells must happen at different places as the testing happens at the end of the last cell of a row, but still inside that cell, so we use two commands. The second is used in the endrow socket.

```

484 \cs_new:Npn \_tbl_store_missing_cells:n #1 {
485   \int_compare:nNnT \g__tbl_col_int > 0
486     {
487     \int_gset:Nn \g__tbl_missing_cells_int
488       {
489         \g__tbl_table_cols_tl
490         - \g__tbl_col_int
491         - \g__tbl_span_tl
492         + 1
493       }
494     \int_compare:nNnT \g__tbl_missing_cells_int < 0 \ERROR % should not happen
495     \typeout{==>~
496       (#1)~
497       This~ row~ needs~
498       \int_use:N \g__tbl_missing_cells_int \space
499       additional~ cell(s)
500     }
501   }
502 }

```

```

503 \cs_new:Npn \_tag_tbl_add_missing_cells:n #1
504   {

```

The TD-socket messages are issued after the message about the end-row socket, but the structure is ok, so better issue a message for now to avoid confusion:

```

505   \int_compare:nNnT {#1}>{0}
506     {
507     \typeout{==>~

```

```

508         ~Inserting~\int_eval:n{#1}~additional~cell(s)~into~previous~row:}
509     }
510     \int_step_inline:nn { #1 }
511     {
512         \UseSocket{tagsupport/tblcell/begin}
513         \UseSocket{tagsupport/tblcell/end}
514     }
515 }

```

(End of definition for `__tbl_store_missing_cells:n` and `__tag_tbl_add_missing_cells:n`.)

`\endarray` If tables are nested into another then it is necessary to restore information about the cell the inner table started in. Otherwise, `\g__tbl_row_int`, `\g__tbl_col_int`, and `\g__tbl_span_tl` reflect the status in the outer table as they are globally manipulated. We restore in all cases even if we are not in a nesting situation as that makes the code simpler (and probably faster).

`\endtabular` and `\endtabular*` inherit from `\endarray` so we only need to change that. `tabularx` is handled below.

```

516 \def\endarray{
517   \__tbl_store_missing_cells:n{endarray}
518   \crrc \egroup
519   \UseSocket{tagsupport/tbl/finalize}
520   \int_gset:Nn \g__tbl_col_int { \l__tbl_saved_col_tl }
521   \int_gset:Nn \g__tbl_row_int { \l__tbl_saved_row_tl }
522   \tl_gset_eq:NN \g__tbl_span_tl \l__tbl_saved_span_tl
523   \tl_gset_eq:NN \g__tbl_table_cols_tl \l__tbl_saved_table_cols_tl
524   \tl_gset_eq:NN \g__tbl_struct_table_tl \l__tbl_saved_struct_table_tl
525   \seq_gset_eq:NN \g__tbl_struct_rows_seq \l__tbl_saved_struct_rows_seq
526   \seq_gset_eq:NN \g__tbl_struct_cells_seq \l__tbl_saved_struct_cells_seq
527   \seq_gset_eq:NN \g__tbl_struct_cur_seq \l__tbl_saved_struct_cur_seq
528   \typeout{==>~ restored~cell~data:~
529             \int_use:N \g__tbl_row_int,
530             \int_use:N \g__tbl_col_int,
531             \l__tbl_saved_span_tl \space
532             (
533             \int_compare:nNnTF \g__tbl_table_cols_tl = 0
534               { outer~ level }
535               { max:~ \g__tbl_table_cols_tl }
536             )
537         }
538   \egroup
539   \@arrayright \gdef\@preamble{}%
540 }

```

(End of definition for `\endarray`. This function is documented on page ??.)

`\@addamp` If we are after the first column we have to insert a `&` and also update the cell data.

```

541 \def\@addamp {
542   \if@firstamp
543     \@firstampfalse
544   \else
545     \edef\@preamble{\@preamble &
546       \__tbl_update_cell_data:
547     }

```

```

548 \fi
549 }

```

(End of definition for \@addamp. This function is documented on page ??.)

```

\g__tbl_col_int \g__tbl_row_int holds the current row number in the table. The value 0 means we
\g__tbl_row_int haven't yet processed the table preamble. It is incremented by every \cr including the
\g__tbl_span_tl one ending the table preamble.
\g__tbl_table_cols_tl \g__tbl_col_int holds the current column number. The value 0 means we have not
yet started the table or just finished a table row (with \\ typically); any other positive
value means we are currently typesetting a cell in that column in some row (denoted by
the \g__tbl_row_int.

```

In a \multicolumn it holds the column number of the first spanned column and \g__tbl_span_tl the info how many cells are spanned.

\g__tbl_span_tl is normally 1 except in a \multicolumn cell.

```

550 \int_new:N \g__tbl_col_int
551 \int_new:N \g__tbl_row_int
552 \tl_new:N \g__tbl_span_tl
553 \tl_new:N \g__tbl_table_cols_tl
554
555 \tl_gset:Nn \g__tbl_span_tl {1}
556 \tl_gset:Nn \g__tbl_table_cols_tl {0} % indicates outer level

```

(End of definition for \g__tbl_col_int and others.)

```

\l__tbl_saved_col_tl Saving the outer values if we are nesting tables is necessary (as the above variables are
\l__tbl_saved_row_tl globally altered. For this we use always token lists because they don't change and we do
\l__tbl_saved_span_tl not need to blow additional integer registers.
\l__tbl_saved_table_cols_tl

```

```

557 \tl_new:N \l__tbl_saved_col_tl
558 \tl_new:N \l__tbl_saved_row_tl
559 \tl_new:N \l__tbl_saved_span_tl
560 \tl_new:N \l__tbl_saved_table_cols_tl
561
562 \tl_set:Nn \l__tbl_saved_col_tl{0}
563 \tl_set:Nn \l__tbl_saved_row_tl{0}
564 \tl_set:Nn \l__tbl_saved_span_tl{1}
565 \tl_set:Nn \l__tbl_saved_table_cols_tl{0} % indicates outer level

```

(End of definition for \l__tbl_saved_col_tl and others.)

```

\g__tbl_struct_table_tl We need to store the structure numbers for the fine tuning in the finalize socket. For now
\l__tbl_saved_struct_table_tl we use a rather simple system: A sequence that hold the numbers for the row structures,
\g__tbl_struct_rows_seq and one that holds comma lists for the cells.
\l__tbl_saved_struct_rows_seq
\g__tbl_struct_cells_seq \g__tbl_struct_table_tl will hold the structure number of the table, \g__tbl_-
\l__tbl_saved_struct_cells_seq struct_rows_seq will hold at index i the structure number of row i, \g__tbl_struct_-
\g__tbl_struct_cur_seq cells_seq will hold at index i a comma list of the cell structure numbers of row i.
\l__tbl_saved_struct_cur_seq \g__tbl_struct_cur_seq is used as a temporary store for the cell structures of the
current row. We need also local version to store and restore the values.

```

```

566 \tl_new:N \g__tbl_struct_table_tl
567 \tl_new:N \l__tbl_saved_struct_table_tl
568 \seq_new:N \g__tbl_struct_rows_seq
569 \seq_new:N \l__tbl_saved_struct_rows_seq
570 \seq_new:N \g__tbl_struct_cells_seq

```

```

571 \seq_new:N \l__tbl_saved_struct_cells_seq
572 \seq_new:N \g__tbl_struct_cur_seq
573 \seq_new:N \l__tbl_saved_struct_cur_seq

```

(End of definition for `\g__tbl_struct_table_tl` and others.)

`\ar@ialign` A new command that replaces `\ialign` above. `\everycr` is also applied to the `\cr` ending the preamble so we have to program around that.

```

574 \def\ar@ialign{%

```

Before starting a table we locally stored the information related to the current cell (if any) so that we can restore it once the table is finished.

```

575 \tl_set:No \l__tbl_saved_col_tl {\int_use:N \g__tbl_col_int }
576 \tl_set:No \l__tbl_saved_row_tl {\int_use:N \g__tbl_row_int }
577 \tl_set_eq:NN \l__tbl_saved_span_tl \g__tbl_span_tl
578 \tl_set_eq:NN \l__tbl_saved_struct_table_tl \g__tbl_struct_table_tl
579 \seq_set_eq:NN \l__tbl_saved_struct_rows_seq \g__tbl_struct_rows_seq
580 \seq_set_eq:NN \l__tbl_saved_struct_cells_seq \g__tbl_struct_cells_seq
581 \seq_set_eq:NN \l__tbl_saved_struct_cur_seq \g__tbl_struct_cur_seq
582 %
583 \typeout{==>~ saved~cell~data:~
584         \l__tbl_saved_row_tl,
585         \l__tbl_saved_col_tl,
586         \l__tbl_saved_span_tl \space
587         (
588         \int_compare:nNnTF \l__tbl_saved_table_cols_tl = 0
589             { outer~ level }
590             { max:~ \l__tbl_saved_table_cols_tl }
591         )
592     }

```

These are the initial values when starting a table:

```

593 \int_gzero:N \g__tbl_row_int
594 \int_gzero:N \g__tbl_col_int
595 \tl_gset:Nn \g__tbl_span_tl {1}
596 \seq_gclear:N\g__tbl_struct_rows_seq
597 \seq_gclear:N\g__tbl_struct_cells_seq
598 \seq_gclear:N\g__tbl_struct_cur_seq
599 \everycr{%
600     \noalign{%

```

We use `\g__tbl_col_int` equal zero to indicate that we are just after a TR (or at the very beginning of the table). Using the row count is not so good as longtable may split the table in chunks.

```

601         \int_compare:nNnT \g__tbl_col_int > 0
602             { \UseSocket{tagsupport/tblrow/end} }
603         \int_gincr:N \g__tbl_row_int           % this row about to start
604         \int_gzero:N \g__tbl_col_int         % we are before first col
605     }%
606 }%
607 \tabskip\z@skip\halign}

```

(End of definition for `\ar@ialign`. This function is documented on page ??.)

`\multicolumn` `\multicolumn` is also defined in `array`. The redefinition has to solve two problems: it must handle the row begin if it is used there, and it must save the numbers of cells it spans so that we can add a suitable `ColSpan` attribute.⁵

```

608 \long\def\multicolumn#1#2#3{%
609 % alternative: determine first col with vmode test ...
610 % \ifvmode
611 %   \multispan{#1}\typeout{A==> vmode}%
612 % \else
613 %   \multispan{#1}\typeout{A==> not vmode}
614 % \fi
615 % but this makes the \crrc handling really complicated which would
616 % then need to become something like
617 %   \ifvmode \expandafter \@gobble
618 %   \else \expandafter \@iden \fi {\cr\noalign{do something}}%
619 % so not used.
620 % Instead:
621   \multispan{#1}\begingroup

```

Insert `rowbegin` socket only if this `multicolumn` replaces the preamble of the first column. In that case we have to set `\g__tbl_col_int` to 1 since this is no longer done in the preamble for the cell.

```

622   \int_compare:NnTF \g__tbl_col_int = 0
623     {
624       \UseSocket{tagsupport/tblrow/begin}
625       \int_gset:Nn \g__tbl_col_int {1}
626     }

```

If we are in a later column we use `\g__tbl_span_tl` from the previous column to update.

```

627     {
628       \int_gadd:Nn \g__tbl_col_int { \g__tbl_span_tl }
629     }

```

Then we set the span value so that it can be use in the next column.

```

630   \tl_gset:Nn \g__tbl_span_tl {#1}
631   \def\@addamp{\if@firstamp\@firstampfalse \else
632     \@preamerr 5\fi}%
633   \@mkpream{#2}\@addtopreamble\@empty
634   \endgroup

```

Now we update the `colspan` attribute. This needs setting after the group as it is hidden inside the plug in `\insert@column`.

```

635   \__tbl_set_colspan:n {#1}
636   \def\@sharp{#3}%
637   \@arstrut \@preamble
638   \null
639   \ignorespaces}

```

(End of definition for `\multicolumn`. This function is documented on page ??.)

⁵FMi: This can now perhaps cleaned up somewhat

5.6 longtable

Longtable is complicated. When at the begin the `\endhead`, `\endfirsthead`, `\endfoot` and `\endlastfoot` are used to setup head and foot they create each a structure subtree with one or more rows. From this structures we want to keep at most two (head and foot) and move the foot to the end of the table. When the head and foot boxes are (re)inserted on following pages we want to mark them up as artifact with the exception of the head at the begin and the foot box at the end.

TODO: When a line is killed the structure subtree is there already too and must be removed.

Hyperref patches longtable. This must be disabled and replace with genuine code

```
640 \let\@kernel@refstepcounter\refstepcounter
641 \def\hyper@nopatch@longtable{}
642 \def\__tbl_patch_LT@array[#1]#2{%
```

`\LT@array` is executed in a group, so we can disable para-tagging here.

```
643 \UseSocket{tagsupport/tbl/init}%
644 \@kernel@refstepcounter{table}\stepcounter{LT@tables}%
```

The target is created rather late and a `\label` can come earlier, so we have to define `\@currentHref` explicitly. We can't currently assume that `\theHtable` is defined always.

```
645 \tl_gset:Ne \@currentHref {table.\cs_if_exist_use:N\theHtable}
646 \int_gzero:N \g__tbl_row_int
647 \seq_gclear:N\g__tbl_struct_rows_seq
648 \seq_gclear:N\g__tbl_struct_cells_seq
649 \seq_gclear:N\g__tbl_struct_cur_seq
650 \seq_gclear:N\g__tbl_LT@firsthead_rows_seq
651 \seq_gclear:N\g__tbl_LT@head_rows_seq
652 \seq_gclear:N\g__tbl_LT@lastfoot_rows_seq
653 \seq_gclear:N\g__tbl_LT@foot_rows_seq
654 \if l#1%
655 \LTleft\z@ \LTright\fill
656 \else\if r#1%
657 \LTleft\fill \LTright\z@
658 \else\if c#1%
659 \LTleft\fill \LTright\fill
660 \fi\fi\fi
661 \let\LT@mc\multicolumn
662 \let\LT@tabarray\@tabarray
663 \let\LT@hline\hline
664 \def\@tabarray{%
665 \let\hline\LT@hline
666 \LT@tabarray}%
667 \let\@LT@tabularcr
668 \let\@tabularnewline\%
669 \def\newpage{\noalign{\break}}%
670 \def\pagebreak{\noalign{\ifnum' }=0\fi\@testopt{\LT@no@pgbk-}4}%
671 \def\nopagebreak{\noalign{\ifnum' }=0\fi\@testopt{\LT@no@pgbk4}}%
672 \let\hline\LT@hline \let\kill\LT@kill\let\caption\LT@caption
673 \@tempdima\ht\strutbox
674 \let\@endpbox\LT@endpbox
675 \ifx\extrarowheight\@undefined
676 \let\@acol\@tabacol
```

```

677 \let\@classz\@tabclassz \let\@classiv\@tabclassiv
678 \def\@startpbox{\vtop\LT@startpbox}%
679 \let\@@startpbox\@startpbox
680 \let\@@endpbox\@endpbox
681 \let\LT@LL@FM@cr\@tabularcr
682 \else
683 \advance\@tempdima\extrarowheight
684 \col@sep\@tabcolsep
685 \let\@startpbox\LT@startpbox\let\LT@LL@FM@cr\@arraycr
686 \fi
687 \setbox\@arstrutbox\hbox{\vrule
688 \@height \arraystretch \@tempdima
689 \@depth \arraystretch \dp \strutbox
690 \@width \z@}%
691 \let\@sharp#\let\protect\relax
692 \begingroup
693 \mkpream{#2}%
694 \tbl_determine_table_cols:
695 \xdef\LT@bchunk{%

```

At the start of a chunk we set `\g__tbl_col_int` to zero to make sure that we aren't generating /TR with the `\cr` ending the chunk preamble.

```

696 \int_gzero:N \g__tbl_col_int
697 \global\advance\c@LT@chunks\@ne
698 \global\LT@rows\z@\setbox\z@\vbox\bgroup
699 \LT@setprevdepth
700 \tabskip\LTleft \noexpand\halign to\hsize\bgroup
701 \tabskip\z@ \@arstrut

```

Insert the socket and the setting of the conditional

```

702 \UseSocket{tagsupport/tblrow/begin}%
703 \tbl_init_cell_data:
704 \@preamble \tabskip\LTright \cr}%
705 \endgroup
706 \expandafter\LT@nofcols\LT@bchunk&\LT@nofcols
707 \LT@make@row
708 \m@th\let\par\@empty

```

Socket and conditional

```

709 \everycr{%
710 \noalign{%

```

In `longtable` we have a bunch of extra `\crs` that are executed whenever a chunk ends. In that case they should not increment the main row counter, sigh.

```

711 \typeout{--longtable-->~chunk~row:~ \the\LT@rows \space
712 row:~ \the\g__tbl_row_int \space
713 column:~ \the\g__tbl_col_int
714 }
715 \int_compare:nNnT \g__tbl_col_int > 0
716 {
717 \UseSocket{tagsupport/tblrow/end}
718 }

```

This prevents any of the additional `\crs` at the end of the chunk to add another /TR. Then once we really start a new chunk it gets incremented so...

```

719 \int_gzero:N \g__tbl_col_int % before first col

```


And for the same reason such `\crs` should not increment the main row counter (but it has to be incremented after the preamble of a chunk), so here we test against `\LT@rows` which is `\LTchunksiz`e at the end of a chunk.

```

720     \int_compare:nNnT \LT@rows < \LTchunksiz
721     { \int_gincr:N \g__tbl_row_int } % this row about to start
722   }%
723 }%

724 \lineskip\z@\baselineskip\z@
725 \LT@bchunk}

```

The end code most stop to insert the endrow too.

```

726 \def\__tbl_patch_endlongtable{%
727   \__tbl_store_missing_cells:n{endlongtable}
728   \crr
729   \noalign{%
730     \UseSocket{tagsupport/tbl/finalize/longtable}
731     \int_gzero:N \g__tbl_row_int % this prevents considering the next
732                                   % \crr as another row end.
733     \let\LT@entry\LT@entry@chop
734     \xdef\LT@save@row{\LT@save@row}}%
735 \LT@echunk
736 \LT@start
737 \unvbox\z@
738 \LT@get@widths
739 \if@filesw
740   {\let\LT@entry\LT@entry@write\immediate\write\@auxout{%
741     \gdef\expandafter\noexpand
742       \csname LT@\romannumeral\c@LT@tables\endcsname
743       {\LT@save@row}}}%
744   \fi
745   \ifx\LT@save@row\LT@save@row
746   \else
747     \LT@warn{Column~\@width s~have~changed\MessageBreak
748             in~table~\thetable}%
749     \LT@final@warn
750   \fi
751   \endgraf\penalty -\LT@end@pen
752   \ifvoid\LT@foot\else
753     \global\advance\vsiz\ht\LT@foot
754     \global\advance\@colroom\ht\LT@foot
755     \dimen@\pagegoal\advance\dimen@\ht\LT@foot\pagegoal\dimen@
756   \fi
757   \endgroup
758   \global\@mparbottom\z@
759   \endgraf\penalty\z@\addvspace\LTpost
760   \ifvoid\footins\else\insert\footins{}\fi}

```

`__tbl_patch_LT@t@bularcr`

```

761 \def\__tbl_patch_LT@t@bularcr{%
762   \global\advance\LT@rows\@ne
763   \ifnum\LT@rows=\LTchunksiz

```

At the end of the chunk `\@` is doing something special and so we loose `__tbl_store_missing_cells:n`. Below is about the right place to add it do this code branch.

```

764   \tbl_store_missing_cells:n{echunk}
765   \gdef\LT@setprevdepth{%
766     \prevdepth\z@
767     \global\let\LT@setprevdepth\relax}%
768   \expandafter\LT@xtabularcr
769   \else
770     \ifnum0='{}\fi
771     \expandafter\LT@LL@FM@cr
772   \fi}
773

```

(End of definition for \tbl_patch_LT@t@bularcr.)

\tbl_patch_LT@end@hd@ft This command is used to store the head and foot boxes. We need to retrieve and store the row so that we can clean up the structure in the finalize code.

```

774 \def\tbl_patch_LT@end@hd@ft#1{%

```

To handle missing columns in the header we need this:

```

775   \tbl_store_missing_cells:n{head/foot}
776   \int_step_inline:nn
777   { \LT@rows + 1 }
778   {
779     \seq_gput_left:ce
780     {g\tbl_\cs_to_str:N #1 _rows_seq }
781     { \int_eval:n {\g\tbl_row_int + 1 - ##1 } }
782   }

```

We also have to set the chunk rows to its max value before calling \LTechunk so that we don't get extra increments of the main row counter due to \everycr.

```

783   \int_gset:Nn \LT@rows { \LTchunksize }
784   \LT@echunk
785   \ifx\LT@start\endgraf
786     \LT@err
787     {Longtable head or foot not at start of table}%
788     {Increase LTchunksize}%
789   \fi
790   \setbox#1\box\z@
791   \LT@get@widths
792   \LT@bchunk}

```

(End of definition for \tbl_patch_LT@end@hd@ft.)

\tbl_patch_LT@start

```

793 \def\tbl_patch_LT@start{%
794   \let\LT@start\endgraf
795   \endgraf\penalty\z@\vskip\LTpre\endgraf
796   \ifdim \pagetotal<\pagegoal \else
797     \dimen@=\pageshrink
798     \advance \dimen@ 1sp %
799     \kern\dimen@\penalty 9999\endgraf \kern-\dimen@
800   \fi
801   \dimen@\pagetotal
802   \advance\dimen@ \ht\ifvoid\LT@firsthead\LT@head\else\LT@firsthead\fi
803   \advance\dimen@ \dp\ifvoid\LT@firsthead\LT@head\else\LT@firsthead\fi
804   \advance\dimen@ \ht\LT@foot

```

```

805 \edef\LT@reset@vfuzz{\vfuzz\the\vfuzz\vbadness\the\vbadness\relax}%
806 \vfuzz\maxdimen
807 \vbadness\@M
808 \setbox\tw@\copy\z@
809 \setbox\tw@\vsplit\tw@ to \ht\@arstrutbox
810 \setbox\tw@\vbox{\unvbox\tw@}%
811 \LT@reset@vfuzz
812 \advance\dimen@ \ht
813 \ifdim\ht\@arstrutbox>\ht\tw@\@arstrutbox\else\tw@\fi
814 \advance\dimen@\dp
815 \ifdim\dp\@arstrutbox>\dp\tw@\@arstrutbox\else\tw@\fi
816 \advance\dimen@ -\pagegoal
817 \ifdim \dimen@>\z@
818 \vfil\break
819 \else
820 \ifdim\pageshrink>\z@\pageshrink\z@\fi
821 \fi
822 \global\@colroom\@colht
823 \ifvoid\LT@foot\else
824 \global\advance\vsizel-\ht\LT@foot
825 \global\advance\@colroom-\ht\LT@foot
826 \dimen@pagegoal\advance\dimen@-\ht\LT@foot\pagegoal\dimen@
827 \maxdepth\z@
828 \fi
829 \MakeLinkTarget{table}
830 \ifvoid\LT@firsthead\copy\LT@head\else\box\LT@firsthead\fi\nobreak

```

Avoid that following uses of the box add content:

```

831 \tagmcbegin{artifact}
832 \tag_mc_reset_box:N\LT@head
833 \tagmccend
834 \output{\LT@output}}

```

(End of definition for _tbl_patch_LT@start.)

_tbl_patch_LT@output We must also avoid that the reuse of the foot box leads to duplicated content:

```

835 \def\_tbl_patch_LT@output{%
836 \ifnum\outputpenalty <-\@Mi
837 \ifnum\outputpenalty > -\LT@endopen
838 \LT@err{floats and marginpars not allowed in a longtable}\@ehc
839 \else
840 \setbox\z@\vbox{\unvbox\@cclv}%
841 \ifdim \ht\LT@lastfoot>\ht\LT@foot
842 \dimen@pagegoal
843 \advance\dimen@\ht\LT@foot
844 \advance\dimen@-\ht\LT@lastfoot
845 \ifdim\dimen@<\ht\z@
846 \setbox\@cclv\vbox{\unvbox\z@\copy\LT@foot\vs}%
847 \@makecol
848 \@outputpage
849 \global\vsizel\@colroom
850 \setbox\z@\vbox{\box\LT@head}%
851 \fi
852 \fi
853 \unvbox\z@\box\ifvoid\LT@lastfoot\LT@foot\else\LT@lastfoot\fi

```

Reset attribute of foot box:

```
854     \tagmcbegin{artifact}
855     \tag_mc_reset_box:N \LT@foot
856     \tagmccend
857     \fi
858   \else
859     \setbox\@cclv\vbox{\unvbox\@cclv\copy\LT@foot\vss}%
```

Reset attribute of foot box:

```
860     \tagmcbegin{artifact}
861     \tag_mc_reset_box:N \LT@foot
862     \tagmccend
863     \@makecol
864     \@outputpage
865     \global\ysize\@colroom
866     \copy\LT@head\nobreak
867     \fi}
```

(End of definition for _tbl_patch_LT@output.)

_tbl_patch_LT@makecaption This patch is quite similar to the one for LaTeX's \@makecaption we also have to change the parbox sockets.

```
868 \def\_tbl_patch_LT@makecaption#1#2#3{%
869   \LT@mc\LT@cols c{%
870     % test can go after merge
871     \str_if_exist:cT { l__socket_tagsupport/parbox/before_plug_str }
872     {
873       \AssignSocketPlug{tagsupport/parbox/before}{noop}
874       \AssignSocketPlug{tagsupport/parbox/after}{noop}
875     }
876     \hbox to\z@{\hss\parbox[t]\LTcapwidth{%
877       \reset@font
878       \tag_stop:n{caption}
879       \sbox\@tempboxa{#1{#2:~}#3}%
880       \tag_start:n{caption}
881       \ifdim\wd\@tempboxa>\hsize
882         #1{#2:~}#3%
883       \else
884         \hbox to\hsize{\hfil#1{#2:~}#3\hfil}%
885       \fi
886       \endgraf\vskip\baselineskip}%
887     \hss}}}
```

(End of definition for _tbl_patch_LT@makecaption.)

Overwrite the longtable definition. That will probably break somewhere as they are various package which patch too.

```
888 \AddToHook{package/longtable/after}
889 {
890   \seq_new:N \g__tbl_LT@firsthead_rows_seq
891   \seq_new:N \g__tbl_LT@head_rows_seq
892   \seq_new:N \g__tbl_LT@lastfoot_rows_seq
893   \seq_new:N \g__tbl_LT@foot_rows_seq
894   \cs_set_eq:NN \LT@array\_tbl_patch_LT@array
895   \cs_set_eq:NN \endlongtable\_tbl_patch_endlongtable
```

```

896 \cs_set_eq:NN \LT@start\_tbl_patch_LT@start
897 \cs_set_eq:NN \LT@output\_tbl_patch_LT@output
898 \cs_set_eq:NN \LT@t@bularcr\_tbl_patch_LT@t@bularcr
899 \cs_set_eq:NN \LT@end@hd@ft\_tbl_patch_LT@end@hd@ft
900 \cs_set_eq:NN \LT@makecaption\_tbl_patch_LT@makecaption
901 }

```

5.7 tabularx

In tabularx we mainly need to ensure that no tagging is done during the trial.

```

902 \def\_tbl_patch_TX@endtabularx{%
903   \expandafter\expandafter\expandafter
904     \TX@find@endtabularxa\csname end\TX@\endcsname
905     \endtabularx\TX@\endtabularx\TX@find@endtabularxa
906   \expandafter\TX@newcol\expandafter{\tabularxcolumn{\TX@col@width}}%
907   \let\verb\TX@verb
908   \def\@elt##1{\global\value{##1}\the\value{##1}\relax}%
909   \edef\TX@ckpt{\cl@ckpt}%
910   \let\@elt\relax
911   \TX@old@table\maxdimen
912   \TX@col@width\TX@target
913   \global\TX@cols@one
914   \TX@typeout@
915     {\@spaces Table Width\@spaces Column Width\@spaces X Columns}%

```

Here we stop tagging:

```

916 \tag_stop:n{tabularx}
917 \TX@trial{\def\NC@rewrite@X{%
918   \global\advance\TX@cols@one\NC@find p{\TX@col@width}}}%
919 \loop
920   \TX@arith
921   \ifTX@
922   \TX@trial{}%
923 \repeat

```

And now we restart it again.

```

924 \tag_start:n{tabularx}
925 {\let\@footnotetext\TX@ftntext\let\@xfootnotenext\TX@xftntext
926   \csname tabular*\expandafter\endcsname\expandafter\TX@target
927   \the\toks@
928   \csname endtabular*\endcsname}%
929 \global\TX@ftn\expandafter{\expandafter}\the\TX@ftn
930 \ifnum0='{fi}%
931 \expandafter\expandafter\expandafter
932 \TX@find@endtabularxbb
933 \expandafter\end\expandafter{\TX@}%
934 \endtabularx\TX@\endtabularx\TX@find@endtabularxb
935 }
936
937 \AddToHook{package/tabularx/after}
938 {\cs_set_eq:NN \TX@endtabularx\_tbl_patch_TX@endtabularx }
939 </package>

```

```
940 <*latex-lab>
941 \ProvidesFile{table-latex-lab-testphase.ltx}
942     [\ltabledate\space v\ltableversion\space latex-lab wrapper table]
943 \RequirePackage{latex-lab-testphase-table}
944 </latex-lab>
```