# The latex-lab-table package
# Changes related to the tagging of tables

Frank & Ulrike, LaTeX 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 longtable . . . . . 23
5.7 tabularx . . . . . 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 table[1] 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 support[2] 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

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

[2] This 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.

[3] While 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

```
⟨@@=tbl⟩
⟨*package⟩
\ProvidesExplPackage {latex-lab-testphase-table} {\ltlabtbldate} {\ltlabtblversion}
  {Code related to the tagging of tables}
```

This builds on array so we load it by default:

```
\RequirePackage{array}
```

[4] Helpful for us at this stage.

## 5.1 Variables

\l__tbl_celltag_tl
\l__tbl_rowtag_tl
\l__tbl_cellattribute_tl
\l__tbl_rowattribute_tl
\g__tbl_missingcells_int
\l__tbl_tmpa_clist
\l__tbl_tmpa_seq
\l__tbl_tmpa_tl

This is for the celltag, e.g. TD or TH:

```
\tl_new:N  \l__tbl_celltag_tl
\tl_set:Nn \l__tbl_celltag_tl {TD}
```

For the rowtag, probably always TR:

```
\tl_new:N  \l__tbl_rowtag_tl
\tl_set:Nn \l__tbl_rowtag_tl {TR}
```

And here cell and row attributes:

```
\tl_new:N  \l__tbl_cellattribute_tl
\tl_set:Nn \l__tbl_cellattribute_tl {}
\tl_new:N  \l__tbl_rowattribute_tl
\tl_set:Nn \l__tbl_rowattribute_tl {}
```

This will contain the number of missing cells used:

```
\int_new:N \g__tbl_missing_cells_int
```

Temp variables

```
\clist_new:N \l__tbl_tmpa_clist
\seq_new:N \l__tbl_tmpa_seq
\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

agsupport/tblcell/begin (*socket*)
tagsupport/tblcell/end (*socket*)
agsupport/tblrow/begin (*socket*)
tagsupport/tblrow/end (*socket*)

```
\NewSocket{tagsupport/tblcell/begin}{0}
\NewSocket{tagsupport/tblcell/end}{0}
\NewSocket{tagsupport/tblrow/begin}{0}
\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.

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

agsupport/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.

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

/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.

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

gsupport/tblhmode/begin (*socket*)
tagsupport/tblhmode/end (*socket*)
gsupport/tblvmode/begin (*socket*)
tagsupport/tblvmode/end (*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.

```
\NewSocket{tagsupport/tblhmode/begin}{0}
\NewSocket{tagsupport/tblhmode/end}{0}
\NewSocket{tagsupport/tblvmode/begin}{0}
\NewSocket{tagsupport/tblvmode/end}{0}
```

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

TD (*plug*)

```
\NewSocketPlug{tagsupport/tblcell/begin}{TD}
  {
    \tag_struct_begin:n
      {
        tag             =\l__tbl_celltag_tl,
        attribute-class =\l__tbl_cellattribute_tl
      }
    \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.

```
    \int_step_inline:nn { \g__tbl_span_tl - 1 }
      {
        \seq_gput_right:Ne \g__tbl_struct_cur_seq { -\tag_get:n {struct_num} }
      }
    \tag_mc_begin:n{}
  }
```

TD (*plug*)

```
\NewSocketPlug{tagsupport/tblcell/end}{TD}
  {
    \tag_mc_end:
    \tag_struct_end:
  }
```

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

TD (*plug*)

```
\NewSocketPlug{tagsupport/tblcell/begin}{TDpbox}
  {
    \tag_struct_begin:n
      {
        tag             =\l__tbl_celltag_tl,
        attribute-class =\l__tbl_cellattribute_tl
      }
    \seq_gput_right:Ne \g__tbl_struct_cur_seq { \tag_get:n {struct_num} }
    \int_step_inline:nn { \g__tbl_span_tl - 1 }
      {
        \seq_gput_right:Ne \g__tbl_struct_cur_seq { -\tag_get:n {struct_num} }
      }
    \tagpdfparaOn
    \tl_set:Nn \l__tag_para_main_tag_tl {Div}
  }
```

TD (*plug*)

```
\NewSocketPlug{tagsupport/tblcell/end}{TDpbox}
  {
    \tag_struct_end:
  }
```

TR (*plug*)

```
\NewSocketPlug{tagsupport/tblrow/begin}{TR}
  {
    \seq_gclear:N \g__tbl_struct_cur_seq
    \tag_struct_begin:n
     {
       tag            =\l__tbl_rowtag_tl,
       attribute-class=\l__tbl_rowattribute_tl
     }
   \seq_gput_right:Ne \g__tbl_struct_rows_seq { \tag_get:n {struct_num} }
  }
```

TR (*plug*)

```
\NewSocketPlug{tagsupport/tblrow/end}{TR}
  {
    \__tag_tbl_add_missing_cells:n { \g__tbl_missing_cells_int }
    \seq_gput_right:Ne \g__tbl_struct_cells_seq
      {
        \seq_use:Nn \g__tbl_struct_cur_seq {,}
      }
    \int_compare:nNnTF { \g__tbl_row_int } = { \seq_count:N\g__tbl_struct_cells_seq }
      {
        \typeout
          {==>~
            stucture~stored~for~row~\int_use:N\g__tbl_row_int :~
            \seq_use:Nn \g__tbl_struct_cur_seq {,}
          }
      }
      { \ERROR } % should not happen ...
    \tag_struct_end:
  }
```

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 reenable paratagging.

```
\NewSocketPlug{tagsupport/tbl/init}{Table}
  {
    \tag_if_active:T
      {
        \bool_set_false:N \l__tag_para_bool
      }
  }
```

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

```
\NewSocketPlug{tagsupport/tbl/finalize}{Table}
```

```
  {
    \__tbl_set_header_rows:
  }
```

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

```
\NewSocketPlug{tagsupport/tbl/finalize/longtable}{Table}
  {
```

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

```
    \bool_lazy_and:nnTF
      { \seq_if_empty_p:N \g__tbl_LT@head_rows_seq  }
      { \seq_if_empty_p:N \g__tbl_LT@firsthead_rows_seq }
      { \__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.

```
      {
        \seq_if_empty:NTF \g__tbl_LT@firsthead_rows_seq
          {
            \clist_set:Ne \l__tbl_header_rows_clist
              {\seq_use:Nn \g__tbl_LT@head_rows_seq {,}}
            \__tbl_set_header_rows:
          }
```

In the other case we use firsthead.

```
          {
            \clist_set:Ne \l__tbl_header_rows_clist
              { \seq_use:Nn \g__tbl_LT@firsthead_rows_seq {,} }
            \__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.

```
            \pdf_version_compare:NnTF < {2.0}
             {
               \seq_map_inline:Nn \g__tbl_LT@head_rows_seq
                 {
                   \seq_gset_item:cnn
                     {g__tag_struct_kids_ \g__tbl_struct_table_tl _seq}
                     { ##1 }
                     {}
```

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.

```
                   \tl_set:Ne \l__tbl_tmpa_tl
                     { \seq_item:Nn\g__tbl_struct_rows_seq {##1} }
                   \prop_if_exist:cT
                     { g__tag_struct_ \l__tbl_tmpa_tl _prop }
                     {
                       %\prop_gremove:cn {g__tag_struct_ \l__tbl_tmpa_tl _prop} {P}
                     }
                 }
```

```
              }
              {
                \seq_map_inline:Nn \g__tbl_LT@head_rows_seq
                  {
                    \tl_set:Ne \l__tbl_tmpa_tl
                      { \seq_item:Nn\g__tbl_struct_rows_seq {##1} }
                    \prop_if_exist:cT
                      { g__tag_struct_ \l__tbl_tmpa_tl _prop }
                      {
                        \__tbl_struct_prop_gput:Vnn \l__tbl_tmpa_tl {S}{/Artifact}
                      }
                  }
              }
            }
        }
```

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.

```
      \bool_lazy_and:nnF
        { \seq_if_empty_p:N \g__tbl_LT@foot_rows_seq  }
        { \seq_if_empty_p:N \g__tbl_LT@lastfoot_rows_seq }
        {
```

If lastfoot is empty move foot to the end.

```
          \seq_if_empty:NTF \g__tbl_LT@lastfoot_rows_seq
            {
              \seq_map_inline:Nn \g__tbl_LT@foot_rows_seq
                {
                  \tl_set:Ne \l__tbl_tmpa_tl
                    {
                      \seq_item:cn
                        {g__tag_struct_kids_ \g__tbl_struct_table_tl _seq}
                        {##1}
                    }
                  \seq_gset_item:cnn
                    {g__tag_struct_kids_ \g__tbl_struct_table_tl _seq}
                    { ##1 }
                    {}
                  \seq_gput_right:cV
                    {g__tag_struct_kids_ \g__tbl_struct_table_tl _seq}
                    \l__tbl_tmpa_tl
                }
            }
```

If lastfoot is not empty we move that.

```
            {
              \seq_map_inline:Nn \g__tbl_LT@lastfoot_rows_seq
                {
                  \tl_set:Ne \l__tbl_tmpa_tl
                    {
                      \seq_item:cn
                        {g__tag_struct_kids_ \g__tbl_struct_table_tl _seq}
                        {##1}
                    }
                  \seq_gset_item:cnn
```

```
                    {g__tag_struct_kids_ \g__tbl_struct_table_tl _seq}
                    { ##1 }
                    {}
                  \seq_gput_right:cV
                    {g__tag_struct_kids_ \g__tbl_struct_table_tl _seq}
                    \l__tbl_tmpa_tl
                }
```

and we hide foot

```
            \pdf_version_compare:NnTF < {2.0}
              {
                \seq_map_inline:Nn \g__tbl_LT@foot_rows_seq
                  {
                    \seq_gset_item:cnn
                      {g__tag_struct_kids_ \g__tbl_struct_table_tl _seq}
                      { ##1 }
                      {}
```

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.

```
                    \tl_set:Ne \l__tbl_tmpa_tl
                      { \seq_item:Nn\g__tbl_struct_rows_seq {##1} }
                    \prop_if_exist:cT
                      { g__tag_struct_ \l__tbl_tmpa_tl _prop }
                      {
                        %\prop_gremove:cn {g__tag_struct_ \l__tbl_tmpa_tl _prop} {P}
                      }
                  }
              }
              {
                \seq_map_inline:Nn \g__tbl_LT@foot_rows_seq
                  {
                    \tl_set:Ne \l__tbl_tmpa_tl
                      { \seq_item:Nn\g__tbl_struct_rows_seq {##1} }
                    \prop_if_exist:cT
                      { g__tag_struct_ \l__tbl_tmpa_tl _prop }
                      {
                        \__tbl_struct_prop_gput:Vnn \l__tbl_tmpa_tl {S}{/Artifact}
                      }
                  }
              }
          }
      }
  }
```

`Table` (*plug*)

```
\NewSocketPlug{tagsupport/tblhmode/begin}{Table}
  {
    \mode_leave_vertical:
    \tag_mc_end_push:
```

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

```
     \bool_lazy_and:nnT
```

```
      { \bool_if_exist_p:N \l__tag_para_bool } { \l__tag_para_bool }
      { \tag_struct_end:n { text } }
    \tag_struct_begin:n {tag=Table}
    \tl_gset:Ne \g__tbl_struct_table_tl { \tag_get:n {struct_num} }
  }
```

Table (*plug*)

```
\NewSocketPlug{tagsupport/tblhmode/end}{Table}
  {
    \tag_struct_end:
```

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

```
    \bool_lazy_and:nnT
      { \bool_if_exist_p:N \l__tag_para_bool } { \l__tag_para_bool }
      { \tag_struct_begin:n { tag=\l__tag_para_tag_tl } }
    \tag_mc_begin_pop:n{}
  }
```

Table (*plug*)

```
\NewSocketPlug{tagsupport/tblvmode/begin}{Table}
  {
    \tag_struct_begin:n {tag=Table}
    \tl_gset:Ne \g__tbl_struct_table_tl { \tag_get:n {struct_num} }
  }
```

Table (*plug*)

```
\NewSocketPlug{tagsupport/tblvmode/end}{Table}
  {
    \tag_struct_end:
    \par
  }
```

## 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!

```
\AddToHook{env/tabular/before}  {\UseSocket{tagsupport/tblhmode/begin}}
\AddToHook{env/tabular/after}   {\UseSocket{tagsupport/tblhmode/end}}
\AddToHook{env/tabular*/before} {\UseSocket{tagsupport/tblhmode/begin}}
\AddToHook{env/tabular*/after}  {\UseSocket{tagsupport/tblhmode/end}}
\AddToHook{env/tabularx/before} {\UseSocket{tagsupport/tblhmode/begin}}
\AddToHook{env/tabularx/after}  {\UseSocket{tagsupport/tblhmode/end}}
\AddToHook{env/longtable/before}{\UseSocket{tagsupport/tblvmode/begin}}
\AddToHook{env/longtable/after} {\UseSocket{tagsupport/tblvmode/end}}
```

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

```
\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.

```
\tag_if_active:T
  { \cs_generate_variant:Nn \__tag_attr_new_entry:nn {ee} }
\cs_new_protected:Npn \__tbl_set_colspan:n #1
  {
    \tag_if_active:T
      {
        \int_compare:nNnT {#1}>{1}
         {
           \prop_get:NeNF \g__tag_attr_entries_prop
              {colspan-\int_eval:n{#1}}
              \l__tbl_tmpa_tl
              {
                \__tag_attr_new_entry:ee
                  {colspan-\int_eval:n{#1}}
                  {/O /Table /ColSpan~\int_eval:n{#1}}
              }
           \tl_set:Ne \l__tbl_cellattribute_tl
              {colspan-\int_eval:n{#1}}
         }
      }
  }
```

(*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:

```
\cs_new_protected:Npn \__tag_tbl_disable:
 {
   \AssignSocketPlug{tagsupport/tblcell/begin}{noop}
   \AssignSocketPlug{tagsupport/tblcell/end}{noop}
   \AssignSocketPlug{tagsupport/tblrow/begin}{noop}
   \AssignSocketPlug{tagsupport/tblrow/end}{noop}
   \AssignSocketPlug{tagsupport/tbl/init}{noop}
   \AssignSocketPlug{tagsupport/tbl/finalize}{noop}
   \AssignSocketPlug{tagsupport/tbl/finalize/longtable}{noop}
   \AssignSocketPlug{tagsupport/tblhmode/begin}{noop}
   \AssignSocketPlug{tagsupport/tblhmode/end}{noop}
   \AssignSocketPlug{tagsupport/tblvmode/begin}{noop}
   \AssignSocketPlug{tagsupport/tblvmode/end}{noop}
 }
```

(*End of definition for* __tag_tbl_disable:.)

__tag_tbl_enable:

```
\cs_new_protected:Npn \__tag_tbl_enable:
 {
   \AssignSocketPlug{tagsupport/tblcell/begin}{TD}
   \AssignSocketPlug{tagsupport/tblcell/end}{TD}
   \AssignSocketPlug{tagsupport/tblrow/begin}{TR}
   \AssignSocketPlug{tagsupport/tblrow/end}{TR}
   \AssignSocketPlug{tagsupport/tbl/init}{Table}
   \AssignSocketPlug{tagsupport/tbl/finalize}{Table}
   \AssignSocketPlug{tagsupport/tbl/finalize/longtable}{Table}
   \AssignSocketPlug{tagsupport/tblhmode/begin}{Table}
   \AssignSocketPlug{tagsupport/tblhmode/end}{Table}
   \AssignSocketPlug{tagsupport/tblvmode/begin}{Table}
   \AssignSocketPlug{tagsupport/tblvmode/end}{Table}
 }
```

(*End of definition for* __tag_tbl_enable:.)

```
% TODO decide about key name
\keys_define:nn { __tag / setup }
  {
    table-tagging .choices:nn = { true, on }
      { \__tag_tbl_enable: },
    table-tagging .choices:nn = { false, off }
      { \__tag_tbl_disable: },
    table-tagging .default:n = true,
    table-tagging .initial:n = true
  }
```

Table tagging should be disabled in the head and foot.

```
\AddToHook{begindocument}
 {
  \cs_if_exist:NT \@kernel@before@head
   {
     \tl_put_right:Nn \@kernel@before@head {\__tag_tbl_disable:}
     \tl_put_right:Nn \@kernel@before@foot {\__tag_tbl_disable:}
   }
 }
```

### 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 `ID`s 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.

```
\AddToHook{begindocument}
 {
  \tag_if_active:T
    {
      \tagpdfsetup
        {
          newattribute =
           {TH-col}{/O /Table /Scope /Column},
          newattribute =
            {TH-row}{/O /Table /Scope /Row},
          newattribute =
            {TH-both}{/O /Table /Scope /Both},
        }
```

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

```
        \seq_gput_left:Ne\g__tag_attr_class_used_seq
          {\pdf_name_from_unicode_e:n{TH-col}}
        \seq_gput_left:Ne\g__tag_attr_class_used_seq
          {\pdf_name_from_unicode_e:n{TH-row}}
        \seq_gput_left:Ne\g__tag_attr_class_used_seq
          {\pdf_name_from_unicode_e:n{TH-both}}
    }
 }

```

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

```
\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:

```
\cs_if_free:NTF \__tag_struct_prop_gput:nnn
  {
    \cs_new_protected:Npn \__tbl_struct_prop_gput:nnn #1#2#3
      { \prop_gput:cnn { g__tag_struct_#1_prop }{#2}{#3} }
  }
  { \cs_new_protected:Npn  \__tbl_struct_prop_gput:nnn #1#2#3
      { \__tag_struct_prop_gput:nnn {#1}{#2}{#3} }
  }
\cs_generate_variant:Nn \__tbl_struct_prop_gput:nnn {nne,Vnn}
\cs_new_protected:Npn \__tbl_set_header_rows:
 {
   \clist_map_inline:Nn \l__tbl_header_rows_clist
     {
       \clist_set:Ne\l__tbl_tmpa_clist
         {
           \seq_item:Nn \g__tbl_struct_cells_seq {##1}
         }
       \clist_map_inline:Nn \l__tbl_tmpa_clist
         {
```

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

```
            \prop_if_exist:cT { g__tag_struct_####1_prop }
              {
                \__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.

```
                \prop_get:cnNTF
                  { g__tag_struct_####1_prop }
                  { C }
                  \l__tbl_tmpa_tl
                  {\__tbl_struct_prop_gput:nne{ ####1 }{C}{[/TH-col~\l__tbl_tmpa_tl]} }
                  {\__tbl_struct_prop_gput:nnn{ ####1 }{C}{/TH-col}}
              }
          }
      }
  }
```

(*End of definition for* `\__tbl_set_header_rows:`.)

And some key support:

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

## 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.

```
\cs_new_protected:Npn \__tbl_show_curr_cell_data: {
   \typeout { ==>~ current~cell~data:~
              \int_use:N \g__tbl_row_int ,
              \int_use:N \g__tbl_col_int ,
              \g__tbl_span_tl
            }
}
```

(*End of definition for* `\__tbl_show_curr_cell_data:`.)

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

```
\def\insert@column{%
  \__tbl_show_curr_cell_data:
  \UseSocket{tagsupport/tblcell/begin}%
  \the@toks \the \@tempcnta
  \ignorespaces \@sharp \unskip
  \the@toks \the \count@ \relax
  \UseSocket{tagsupport/tblcell/end}%
}
```

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

\@classz

```
\def\@classz{\@classx
   \@tempcnta \count@
   \prepnext@tok
   \@addtopreamble{\ifcase \@chnum
      \hfil
      \hskip1sp%
      \d@llarbegin
      \insert@column
      \d@llarend \do@row@strut \hfil \or
      \hskip1sp\d@llarbegin \insert@column \d@llarend \do@row@strut \hfil \or
      \hfil\hskip1sp\d@llarbegin \insert@column \d@llarend \do@row@strut \or
    \setbox\ar@mcellbox\vbox
    \@startpbox{\@nextchar}
    \AssignSocketPlug{tagsupport/tblcell/begin}{TDpbox}
    \AssignSocketPlug{tagsupport/tblcell/end}{TDpbox}
    \insert@column \@endpbox
    \ar@align@mcell
    \do@row@strut \or
   \vtop \@startpbox{\@nextchar}
    \AssignSocketPlug{tagsupport/tblcell/begin}{TDpbox}
    \AssignSocketPlug{tagsupport/tblcell/end}{TDpbox}
    \insert@column \@endpbox\do@row@strut \or
   \vbox \@startpbox{\@nextchar}
    \AssignSocketPlug{tagsupport/tblcell/begin}{TDpbox}
    \AssignSocketPlug{tagsupport/tblcell/end}{TDpbox}
    \insert@column \@endpbox\do@row@strut
  \fi}\prepnext@tok}
```

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

\@array
\@@array

We modificate the `\@array` from array.

```
\def\@array[#1]#2{%
  \@tempdima \ht \strutbox
  \advance \@tempdima by\extrarowheight
  \setbox \@arstrutbox \hbox{\vrule
             \@height \arraystretch \@tempdima
             \@depth \arraystretch \dp \strutbox
             \@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.

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

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

```
  \__tbl_determine_table_cols:
  \xdef\@preamble{%
    \noexpand
```

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

```
      \ar@ialign
      \@halignto
      \bgroup \@arstrut
```

A socket is inserted

```
       \UseSocket{tagsupport/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:.

```
       \__tbl_init_cell_data:
       \@preamble
       \tabskip \z@ \cr}%
  \endgroup
  \@arrayleft
```

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

```
  \UseSocket{tagsupport/tbl/init}
  \if #1t\vtop \else \if#1b\vbox \else \vcenter \fi \fi
  \bgroup
  \let \@sharp ##\let \protect \relax
  \lineskip \z@
  \baselineskip \z@
  \m@th
  \let\\\@arraycr \let\tabularnewline\\\let\par\@empty
%\show\@preamble
  \@preamble}
```

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

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

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

__tbl_init_cell_data:

```
\cs_new_protected:Npn \__tbl_init_cell_data: {
        \int_gset:Nn \g__tbl_col_int {1}
        \tl_gset:Nn  \g__tbl_span_tl {1}
}
```

(*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).

```
\cs_new_protected:Npn \__tbl_update_cell_data: {
        \int_gadd:Nn \g__tbl_col_int { \g__tbl_span_tl }
        \tl_gset:Nn  \g__tbl_span_tl {1}
}
```

(*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.

```
\cs_new:Npn \__tbl_determine_table_cols:  {
  \seq_set_split:NnV\l__tbl_tmpa_seq {&}\@preamble
  \tl_gset:Ne \g__tbl_table_cols_tl { \seq_count:N \l__tbl_tmpa_seq }
  \typeout{ ==>~ Table~ has~ \g__tbl_table_cols_tl \space columns }
}
```

(*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.

```
\protected\def\@arraycr{
  \relax
  \__tbl_store_missing_cells:n{@arraycr}
  %
  \iffalse{\fi\ifnum 0=`}\fi
  \@ifstar \@xarraycr \@xarraycr}
```

(*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.

```
\cs_new:Npn \__tbl_store_missing_cells:n #1 {
  \int_compare:nNnT \g__tbl_col_int > 0
      {
        \int_gset:Nn \g__tbl_missing_cells_int
            {
              \g__tbl_table_cols_tl
            - \g__tbl_col_int
            - \g__tbl_span_tl
            + 1
            }
        \int_compare:nNnT \g__tbl_missing_cells_int < 0 \ERROR % should not happen
        \typeout{==>~
          (#1)~
          This~ row~ needs~
          \int_use:N \g__tbl_missing_cells_int \space
          additional~ cell(s)
        }
      }
}
\cs_new:Npn \__tag_tbl_add_missing_cells:n #1
  {
```

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:

```
    \int_compare:nNnT {#1}>{0}
      {
       \typeout{==>~
```

```
            ~Inserting~\int_eval:n{#1}~additional~cell(s)~into~previous~row:}
        }
      \int_step_inline:nn { #1 }
        {
          \UseSocket{tagsupport/tblcell/begin}
          \UseSocket{tagsupport/tblcell/end}
        }
  }
```

(*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.

```
\def\endarray{
  \__tbl_store_missing_cells:n{endarray}
  \crcr \egroup
  \UseSocket{tagsupport/tbl/finalize}
  \int_gset:Nn \g__tbl_col_int { \l__tbl_saved_col_tl }
  \int_gset:Nn \g__tbl_row_int { \l__tbl_saved_row_tl }
  \tl_gset_eq:NN \g__tbl_span_tl \l__tbl_saved_span_tl
  \tl_gset_eq:NN \g__tbl_table_cols_tl   \l__tbl_saved_table_cols_tl
  \tl_gset_eq:NN \g__tbl_struct_table_tl  \l__tbl_saved_struct_table_tl
  \seq_gset_eq:NN \g__tbl_struct_rows_seq \l__tbl_saved_struct_rows_seq
  \seq_gset_eq:NN \g__tbl_struct_cells_seq\l__tbl_saved_struct_cells_seq
  \seq_gset_eq:NN \g__tbl_struct_cur_seq  \l__tbl_saved_struct_cur_seq
  \typeout{==>~ restored~cell~data:~
                \int_use:N \g__tbl_row_int,
                \int_use:N \g__tbl_col_int,
                \l__tbl_saved_span_tl \space
                (
                \int_compare:nNnTF \g__tbl_table_cols_tl = 0
                    { outer~ level }
                    { max:~ \g__tbl_table_cols_tl }
                )
          }
  \egroup
  \@arrayright \gdef\@preamble{}%
}
```

(*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.

```
\def\@addamp {
  \if@firstamp
    \@firstampfalse
  \else
    \edef\@preamble{\@preamble &
        \__tbl_update_cell_data:
    }
```

```
  \fi
}
```

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

`\g__tbl_col_int`
`\g__tbl_row_int`
`\g__tbl_span_tl`
`\g__tbl_table_cols_tl`

`\g__tbl_row_int` holds the current row number in the table. The value `0` means we haven't yet processed the table preamble. It is incremented by every `\cr` including the one ending the table preamble.

`\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.

```
\int_new:N \g__tbl_col_int
\int_new:N \g__tbl_row_int
\tl_new:N  \g__tbl_span_tl
\tl_new:N  \g__tbl_table_cols_tl

\tl_gset:Nn \g__tbl_span_tl {1}
\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`
`\l__tbl_saved_row_tl`
`\l__tbl_saved_span_tl`
`\l__tbl_saved_table_cols_tl`

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

```
\tl_new:N \l__tbl_saved_col_tl
\tl_new:N \l__tbl_saved_row_tl
\tl_new:N \l__tbl_saved_span_tl
\tl_new:N \l__tbl_saved_table_cols_tl

\tl_set:Nn \l__tbl_saved_col_tl{0}
\tl_set:Nn \l__tbl_saved_row_tl{0}
\tl_set:Nn \l__tbl_saved_span_tl{1}
\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`
`\l__tbl_saved_struct_table_tl`
`\g__tbl_struct_rows_seq`
`\l__tbl_saved_struct_rows_seq`
`\g__tbl_struct_cells_seq`
`\l__tbl_saved_struct_cells_seq`
`\g__tbl_struct_cur_seq`
`\l__tbl_saved_struct_cur_seq`

We need to store the structure numbers for the fine tuning in the finalize socket. For now we use a rather simple system: A sequence that hold the numbers for the row structures, and one that holds comma lists for the cells.

`\g__tbl_struct_table_tl` will hold the structure number of the table, `\g__tbl_struct_rows_seq` will hold at index i the structure number of row i, `\g__tbl_struct_cells_seq` will hold at index i a comma list of the cell structure numbers of row i. `\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.

```
\tl_new:N  \g__tbl_struct_table_tl
\tl_new:N  \l__tbl_saved_struct_table_tl
\seq_new:N \g__tbl_struct_rows_seq
\seq_new:N \l__tbl_saved_struct_rows_seq
\seq_new:N \g__tbl_struct_cells_seq
```

```
\seq_new:N \l__tbl_saved_struct_cells_seq
\seq_new:N \g__tbl_struct_cur_seq
\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.

```
\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.

```
  \tl_set:No \l__tbl_saved_col_tl {\int_use:N \g__tbl_col_int }
  \tl_set:No \l__tbl_saved_row_tl {\int_use:N \g__tbl_row_int }
  \tl_set_eq:NN \l__tbl_saved_span_tl  \g__tbl_span_tl
  \tl_set_eq:NN  \l__tbl_saved_struct_table_tl \g__tbl_struct_table_tl
  \seq_set_eq:NN \l__tbl_saved_struct_rows_seq \g__tbl_struct_rows_seq
  \seq_set_eq:NN \l__tbl_saved_struct_cells_seq \g__tbl_struct_cells_seq
  \seq_set_eq:NN \l__tbl_saved_struct_cur_seq \g__tbl_struct_cur_seq
%
  \typeout{==>~ saved~cell~data:~
                \l__tbl_saved_row_tl,
                \l__tbl_saved_col_tl,
                \l__tbl_saved_span_tl \space
                (
                \int_compare:nNnTF \l__tbl_saved_table_cols_tl = 0
                    { outer~ level }
                    { max:~ \l__tbl_saved_table_cols_tl }
                )
          }
```

These are the initial values when starting a table:

```
  \int_gzero:N \g__tbl_row_int
  \int_gzero:N \g__tbl_col_int
  \tl_gset:Nn  \g__tbl_span_tl {1}
  \seq_gclear:N\g__tbl_struct_rows_seq
  \seq_gclear:N\g__tbl_struct_cells_seq
  \seq_gclear:N\g__tbl_struct_cur_seq

  \everycr{%
    \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.

```
      \int_compare:nNnT \g__tbl_col_int > 0
         { \UseSocket{tagsupport/tblrow/end} }
      \int_gincr:N \g__tbl_row_int          % this row about to start
      \int_gzero:N \g__tbl_col_int          % we are before first col
    }%
  }%
  \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]

```
\long\def\multicolumn#1#2#3{%
  % alternative: determine first col with vmode test ...
  %  \ifvmode
  %    \multispan{#1}\typeout{A==> vmode}%
  %  \else
  %    \multispan{#1}\typeout{A==> not vmode}
  %  \fi
  % but this makes the \crcr handling really complicated which would
  % then need to become something like
  %    \ifvmode \expandafter \@gobble
  %    \else \expandafter \@iden \fi {\cr\noalign{do something}}%
  % so not used.
  % Instead:
   \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.

```
    \int_compare:nNnTF \g__tbl_col_int = 0
        {
          \UseSocket{tagsupport/tblrow/begin}
          \int_gset:Nn \g__tbl_col_int {1}
        }
```

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

```
        {
          \int_gadd:Nn \g__tbl_col_int { \g__tbl_span_tl }
        }
```

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

```
    \tl_gset:Nn \g__tbl_span_tl {#1}

   \def\@addamp{\if@firstamp\@firstampfalse \else
                  \@preamerr 5\fi}%
    \@mkpream{#2}\@addtopreamble\@empty
    \endgroup
```

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

```
    \__tbl_set_colspan:n {#1}

    \def\@sharp{#3}%
    \@arstrut \@preamble
    \null
    \ignorespaces}
```

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

[5] FMi: 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

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

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

```
  \UseSocket{tagsupport/tbl/init}%
  \@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.

```
  \tl_gset:Ne \@currentHref {table.\cs_if_exist_use:N\theHtable}
  \int_gzero:N \g__tbl_row_int
  \seq_gclear:N\g__tbl_struct_rows_seq
  \seq_gclear:N\g__tbl_struct_cells_seq
  \seq_gclear:N\g__tbl_struct_cur_seq
  \seq_gclear:N\g__tbl_LT@firsthead_rows_seq
  \seq_gclear:N\g__tbl_LT@head_rows_seq
  \seq_gclear:N\g__tbl_LT@lastfoot_rows_seq
  \seq_gclear:N\g__tbl_LT@foot_rows_seq
  \if l#1%
    \LTleft\z@ \LTright\fill
  \else\if r#1%
    \LTleft\fill \LTright\z@
  \else\if c#1%
    \LTleft\fill \LTright\fill
  \fi\fi\fi
  \let\LT@mcol\multicolumn
  \let\LT@@tabarray\@tabarray
  \let\LT@@hl\hline
  \def\@tabarray{%
    \let\hline\LT@@hl
    \LT@@tabarray}%
  \let\\\LT@tabularcr
  \let\tabularnewline\\%
  \def\newpage{\noalign{\break}}%
  \def\pagebreak{\noalign{\ifnum`}=0\fi\@testopt{\LT@no@pgbk-}4}%
  \def\nopagebreak{\noalign{\ifnum`}=0\fi\@testopt\LT@no@pgbk4}%
  \let\hline\LT@hline \let\kill\LT@kill\let\caption\LT@caption
  \@tempdima\ht\strutbox
  \let\@endpbox\LT@endpbox
  \ifx\extrarowheight\@undefined
    \let\@acol\@tabacol
```

```
    \let\@classz\@tabclassz \let\@classiv\@tabclassiv
    \def\@startpbox{\vtop\LT@startpbox}%
    \let\@@startpbox\@startpbox
    \let\@@endpbox\@endpbox
    \let\LT@LL@FM@cr\@tabularcr
  \else
    \advance\@tempdima\extrarowheight
    \col@sep\tabcolsep
    \let\@startpbox\LT@startpbox\let\LT@LL@FM@cr\@arraycr
  \fi
  \setbox\@arstrutbox\hbox{\vrule
    \@height \arraystretch \@tempdima
    \@depth \arraystretch \dp \strutbox
    \@width \z@}%
  \let\@sharp##\let\protect\relax
   \begingroup
    \@mkpream{#2}%
    \__tbl_determine_table_cols:
    \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.

```
      \int_gzero:N \g__tbl_col_int
      \global\advance\c@LT@chunks\@ne
      \global\LT@rows\z@\setbox\z@\vbox\bgroup
      \LT@setprevdepth
      \tabskip\LTleft \noexpand\halign to\hsize\bgroup
      \tabskip\z@ \@arstrut
```

Insert the socket and the setting of the conditional

```
      \UseSocket{tagsupport/tblrow/begin}%
      \__tbl_init_cell_data:
      \@preamble \tabskip\LTright \cr}%
  \endgroup
  \expandafter\LT@nofcols\LT@bchunk&\LT@nofcols
  \LT@make@row
  \m@th\let\par\@empty
```

Socket and conditional

```
  \everycr{%
    \noalign{%
```

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

```
      \typeout{--longtable-->~chunk~row:~ \the\LT@rows \space
               row:~ \the\g__tbl_row_int   \space
               column:~ \the\g__tbl_col_int
      }
      \int_compare:nNnT \g__tbl_col_int > 0
          {
            \UseSocket{tagsupport/tblrow/end}
          }
```

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

```
      \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 `\LTchunksize` at the end of a chunk.

```
      \int_compare:nNnT \LT@rows < \LTchunksize
         { \int_gincr:N \g__tbl_row_int  }   % this row about to start
    }%
  }%
  \lineskip\z@\baselineskip\z@
  \LT@bchunk}
```

The end code most stop to insert the endrow too.

```
\def\__tbl_patch_endlongtable{%
  \__tbl_store_missing_cells:n{endlongtable}
  \crcr
  \noalign{%
    \UseSocket{tagsupport/tbl/finalize/longtable}
    \int_gzero:N \g__tbl_row_int      % this prevents considering the next
                                      % \crcr as another row end.
    \let\LT@entry\LT@entry@chop
    \xdef\LT@save@row{\LT@save@row}}%
  \LT@echunk
  \LT@start
  \unvbox\z@
  \LT@get@widths
  \if@filesw
    {\let\LT@entry\LT@entry@write\immediate\write\@auxout{%
      \gdef\expandafter\noexpand
        \csname LT@\romannumeral\c@LT@tables\endcsname
          {\LT@save@row}}}%
  \fi
  \ifx\LT@save@row\LT@@save@row
  \else
    \LT@warn{Column~\@width s~have~changed\MessageBreak
             in~table~\thetable}%
    \LT@final@warn
  \fi
  \endgraf\penalty -\LT@end@pen
  \ifvoid\LT@foot\else
    \global\advance\vsize\ht\LT@foot
    \global\advance\@colroom\ht\LT@foot
    \dimen@\pagegoal\advance\dimen@\ht\LT@foot\pagegoal\dimen@
  \fi
  \endgroup
  \global\@mparbottom\z@
  \endgraf\penalty\z@\addvspace\LTpost
  \ifvoid\footins\else\insert\footins{}\fi}
```

`\__tbl_patch_LT@t@bularcr`

```
\def\__tbl_patch_LT@t@bularcr{%
  \global\advance\LT@rows\@ne
  \ifnum\LT@rows=\LTchunksize
```

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.

```
    \__tbl_store_missing_cells:n{echunk}
    \gdef\LT@setprevdepth{%
      \prevdepth\z@
      \global\let\LT@setprevdepth\relax}%
    \expandafter\LT@xtabularcr
  \else
    \ifnum0=`{}\fi
    \expandafter\LT@LL@FM@cr
  \fi}

```

(*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.

```
\def\__tbl_patch_LT@end@hd@ft#1{%
```

To handle missing columns in the header we need this:

```
  \__tbl_store_missing_cells:n{head/foot}
  \int_step_inline:nn
   { \LT@rows + 1 }
   {
     \seq_gput_left:ce
       {g__tbl_\cs_to_str:N #1 _rows_seq }
       { \int_eval:n {\g__tbl_row_int + 1 - ##1 } }
   }
```

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`.

```
  \int_gset:Nn \LT@rows { \LTchunksize }
  \LT@echunk
  \ifx\LT@start\endgraf
    \LT@err
     {Longtable head or foot not at start of table}%
     {Increase LTchunksize}%
  \fi
  \setbox#1\box\z@
  \LT@get@widths
  \LT@bchunk}
```

(*End of definition for* `\__tbl_patch_LT@end@hd@ft`.)

`\__tbl_patch_LT@start`

```
\def\__tbl_patch_LT@start{%
  \let\LT@start\endgraf
  \endgraf\penalty\z@\vskip\LTpre\endgraf
   \ifdim \pagetotal<\pagegoal \else
      \dimen@=\pageshrink
      \advance \dimen@ 1sp %
      \kern\dimen@\penalty 9999\endgraf \kern-\dimen@
   \fi
  \dimen@\pagetotal
  \advance\dimen@ \ht\ifvoid\LT@firsthead\LT@head\else\LT@firsthead\fi
  \advance\dimen@ \dp\ifvoid\LT@firsthead\LT@head\else\LT@firsthead\fi
  \advance\dimen@ \ht\LT@foot
```

```
\edef\LT@reset@vfuzz{\vfuzz\the\vfuzz\vbadness\the\vbadness\relax}%
\vfuzz\maxdimen
\vbadness\@M
\setbox\tw@\copy\z@
\setbox\tw@\vsplit\tw@ to \ht\@arstrutbox
\setbox\tw@\vbox{\unvbox\tw@}%
\LT@reset@vfuzz
\advance\dimen@ \ht
      \ifdim\ht\@arstrutbox>\ht\tw@\@arstrutbox\else\tw@\fi
\advance\dimen@\dp
      \ifdim\dp\@arstrutbox>\dp\tw@\@arstrutbox\else\tw@\fi
\advance\dimen@ -\pagegoal
\ifdim \dimen@>\z@
  \vfil\break
\else
  \ifdim\pageshrink>\z@\pageshrink\z@\fi
\fi
    \global\@colroom\@colht
\ifvoid\LT@foot\else
  \global\advance\vsize-\ht\LT@foot
  \global\advance\@colroom-\ht\LT@foot
  \dimen@\pagegoal\advance\dimen@-\ht\LT@foot\pagegoal\dimen@
  \maxdepth\z@
\fi
\MakeLinkTarget{table}
\ifvoid\LT@firsthead\copy\LT@head\else\box\LT@firsthead\fi\nobreak
```

Avoid that following uses of the box add content:

```
\tagmcbegin{artifact}
 \tag_mc_reset_box:N\LT@head
\tagmcend
\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:

```
\def\__tbl_patch_LT@output{%
  \ifnum\outputpenalty <-\@Mi
    \ifnum\outputpenalty > -\LT@end@pen
      \LT@err{floats and marginpars not allowed in a longtable}\@ehc
    \else
      \setbox\z@\vbox{\unvbox\@cclv}%
      \ifdim \ht\LT@lastfoot>\ht\LT@foot
        \dimen@\pagegoal
        \advance\dimen@\ht\LT@foot
        \advance\dimen@-\ht\LT@lastfoot
        \ifdim\dimen@<\ht\z@
          \setbox\@cclv\vbox{\unvbox\z@\copy\LT@foot\vss}%
          \@makecol
          \@outputpage
          \global\vsize\@colroom
          \setbox\z@\vbox{\box\LT@head}%
        \fi
      \fi
        \unvbox\z@\box\ifvoid\LT@lastfoot\LT@foot\else\LT@lastfoot\fi
```

Reset attribute of foot box:

```
        \tagmcbegin{artifact}
        \tag_mc_reset_box:N \LT@foot
        \tagmcend
    \fi
  \else
    \setbox\@cclv\vbox{\unvbox\@cclv\copy\LT@foot\vss}%
```

Reset attribute of foot box:

```
    \tagmcbegin{artifact}
    \tag_mc_reset_box:N \LT@foot
    \tagmcend
    \@makecol
    \@outputpage
      \global\vsize\@colroom
    \copy\LT@head\nobreak
  \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.

```
\def\__tbl_patch_LT@makecaption#1#2#3{%
  \LT@mcol\LT@cols c{%
  % test can go after merge
    \str_if_exist:cT { l__socket_tagsupport/parbox/before_plug_str }
      {
        \AssignSocketPlug{tagsupport/parbox/before}{noop}
        \AssignSocketPlug{tagsupport/parbox/after}{noop}
      }
    \hbox to\z@{\hss\parbox[t]\LTcapwidth{%
    \reset@font
    \tag_stop:n{caption}
    \sbox\@tempboxa{#1{#2:~}#3}%
    \tag_start:n{caption}
    \ifdim\wd\@tempboxa>\hsize
      #1{#2:~}#3%
    \else
      \hbox to\hsize{\hfil#1{#2:~}#3\hfil}%
    \fi
    \endgraf\vskip\baselineskip}%
  \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.

```
\AddToHook{package/longtable/after}
  {
   \seq_new:N \g__tbl_LT@firsthead_rows_seq
   \seq_new:N \g__tbl_LT@head_rows_seq
   \seq_new:N \g__tbl_LT@lastfoot_rows_seq
   \seq_new:N \g__tbl_LT@foot_rows_seq
   \cs_set_eq:NN \LT@array\__tbl_patch_LT@array
   \cs_set_eq:NN \endlongtable\__tbl_patch_endlongtable
```

```
  \cs_set_eq:NN \LT@start\__tbl_patch_LT@start
  \cs_set_eq:NN \LT@output\__tbl_patch_LT@output
  \cs_set_eq:NN \LT@t@bularcr\__tbl_patch_LT@t@bularcr
  \cs_set_eq:NN \LT@end@hd@ft\__tbl_patch_LT@end@hd@ft
  \cs_set_eq:NN \LT@makecaption\__tbl_patch_LT@makecaption
 }
```

## 5.7 tabularx

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

```
\def\__tbl_patch_TX@endtabularx{%
   \expandafter\expandafter\expandafter
     \TX@find@endtabularxa\csname end\TX@\endcsname
     \endtabularx\TX@\endtabularx\TX@find@endtabularxa
  \expandafter\TX@newcol\expandafter{\tabularxcolumn{\TX@col@width}}%
  \let\verb\TX@verb
  \def\@elt##1{\global\value{##1}\the\value{##1}\relax}%
  \edef\TX@ckpt{\cl@@ckpt}%
  \let\@elt\relax
  \TX@old@table\maxdimen
  \TX@col@width\TX@target
  \global\TX@cols\@ne
  \TX@typeout@
    {\@spaces Table Width\@spaces Column Width\@spaces X Columns}%
```

Here we stop tagging:

```
  \tag_stop:n{tabularx}
  \TX@trial{\def\NC@rewrite@X{%
          \global\advance\TX@cols\@ne\NC@find p{\TX@col@width}}}%
  \loop
    \TX@arith
    \ifTX@
    \TX@trial{}%
  \repeat
```

And now we restart it again.

```
  \tag_start:n{tabularx}
  {\let\@footnotetext\TX@ftntext\let\@xfootnotenext\TX@xftntext
    \csname tabular*\expandafter\endcsname\expandafter\TX@target
      \the\toks@
    \csname endtabular*\endcsname}%
  \global\TX@ftn\expandafter{\expandafter}\the\TX@ftn
  \ifnum0=‘{\fi}%
   \expandafter\expandafter\expandafter
   \TX@find@endtabularxbb
    \expandafter\end\expandafter{\TX@}%
    \endtabularx\TX@\endtabularx\TX@find@endtabularxb
}

\AddToHook{package/tabularx/after}
  {\cs_set_eq:NN \TX@endtabularx\__tbl_patch_TX@endtabularx }
```

⟨/package⟩

```
⟨*latex-lab⟩
\ProvidesFile{table-latex-lab-testphase.ltx}
        [\ltlabtbldate\space v\ltlabtblversion\space latex-lab wrapper table]
\RequirePackage{latex-lab-testphase-table}
⟨/latex-lab⟩
```