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

Frank & Ulrike, LaTeX Project*

v0.85v 2026-09-24

Abstract

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

Contents

*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.

From a high level perspective the tagging of tables 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 structure. Headers are only produced through visual formatting, e.g., by making them bold or by underlying some color. But accessible tables should have properly marked up headers 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 implements only some basic support for the tagging of tables. A list of the known limitations is shown below.

1.1 Use

The module is loaded when you use `\DocumentMetadata` and if tagging is active it will automatically tag all already supported table environments the exception of tables in the header and footer of the page (where tagging is disabled). Such tables can be nested.

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.

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.[2]

As mentioned below the colortbl doesn't work fully with the tagging, but when it does, colors inside the table are currently ignored by the tagging. If such a color has a semantic meaning (like "important value") this meaning will be lost.

1.2 Untagged and presentation tables

If a table should not be tagged as table, for example because it is merely used as a layout tool to ensure that the content is properly aligned or because it is a not yet (fully) supported table structure, the tagging can be disabled with `\tagpdfsetup{table/tagging=false}`

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

[2] 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.

or changed with `\tagpdfsetup{table/tagging=presentation}` or `\tagpdfsetup{table/tagging=div`[3] The first option disables the table tagging code and the content of the tabular is then treated more or less like running text. This works ok for simple tables using only hmode-cells (l/c/r) with normal text content, but fails if the table uses vmode-cells (p/m/b). In such cases the second option works better: it keeps the tagging code active, but adds an ARIA-role as attribute to indicate that this is a presentation table. When deriving to html this gives the best result as it keeps the layout. The last option changes the tag names and creates simply a number of DIV structures. Both options also (re)set the `table/header-rows` and `table/header-columns` keys to empty.

To reset to the normal tagging in the current group (e.g. inside a presentation table) one can use `\tagpdfsetup{table/tagging=true}` (and if needed reenable the headers).

1.3 Header rows and columns

There is some basic support[4] for headers. With

```
\tagpdfsetup{table/header-rows={⟨list of row numbers⟩}}
\tagpdfsetup{table/header-columns={⟨list of column numbers⟩}}
```

you can declare which (absolute) row and column numbers should be tagged as header rows and header columns. It applies to all tables until it is changed to a different list of row or column numbers or undone by setting the keys to ⟨*empty*⟩. A row or column number can be negative, then the counting starts from the end of the table. In a `longtable` the code will 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.

1.4 Spanning of cells

`\multicolumn` is supported out of the box and will create a structure with a suitable `ColSpan` attribute[5]

For cells spanning rows some preliminary support exists[6]: If you add

```
\tagpdfsetup{table/multirow={⟨number of rows⟩}}
```

to a cell the code will add the suitable `RowSpan` attribute and suppress the tagging of affected cells in the following rows. This will also work if the current cell is a `\multicolumn`, then the cell will span both rows and columns. It is the duty of the author to ensure that all cells below and covered by a `RowSpan` are empty! The code neither checks that nor does it tries to suppress content.

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.

[3] The value `presentation` refers to the ARIA role "presentation".

[4] This is not meant to be the final interface, though.

[5] The code uses actually an attribute *class*. The validator PAC doesn't handle this correctly currently and complains about a missing attribute.

[6] The interface is bound to change!

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 can only be added manually (the multirow package is untested).
- The colortbl package breaks tagging if there are nested tables.
- The tabularray package uses a completely different method to create tables and is not supported by this code.
- Tables in text mode made with nicematrix package are often correctly tagged, but should be carefully checked. In math mode they are currently not compatible with the MathML structure elements.
- Most other packages related to tables in LaTeX are not fully tested; that includes packages that change rules like booktabs, hhline, arydshln, hvdashln. Some problems have been resolved, either in the packages or through a firstaid.
- `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 title pages. In such uses the tagging of the table must be deactivated with `\tagpdfsetup{table/tagging=false}` or changed with `\tagpdfsetup{table/tagging=presentation}` or `\tagpdfsetup{table/tagging=div}`.
- Only simple header rows and columns are currently supported. Complex headers with subheaders will be handled later as that needs some syntax changes. Tables with more than one header row or column are probably not pdf/UA conformant as the headers array in the cells is missing.
- 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.
- Vertical boxes (`\parbox`, `minipage`, …) inside cells can be problematic.
- The code is quite noisy and fills the log with lots of messages.[7]

[7] Helpful for us at this stage.

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.
- More packages must be tested.

5 Implementation

```
1 ⟨@@=tbl⟩
2 ⟨*package⟩

3 \ProvidesExplPackage {latex-lab-testphase-table} {\ltlabtbldate} {\ltlabtblversion}
4   {Code related to the tagging of tables}
```

This builds on array so we load it by default:

```
5 \RequirePackage{array}
```

5.1 Variables

\l__tbl_celltag_tl \l__tbl_pcelltag_tl
\l__tbl_rowtag_tl
\l__tbl_table_tl
\l__tbl_cellattribute_tl
\l__tbl_rowattribute_tl
\l__tbl_tmpa_clist
\l__tbl_tmpa_tl
\l__tbl_tmpa_str

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

```
6 \tl_new:N  \l__tbl_celltag_tl
7 \tl_set:Nn \l__tbl_celltag_tl {\UseStructureName{table/cell}}
8 \tl_new:N  \l__tbl_pcelltag_tl
9 \tl_set:Nn \l__tbl_pcelltag_tl {\UseStructureName{table/cell}}
```

For the rowtag, probably always TR:

```
10 \tl_new:N  \l__tbl_rowtag_tl
11 \tl_set:Nn \l__tbl_rowtag_tl {\UseStructureName{table/row}}
```

For the tabletag, probably always Table:

```
12 \tl_new:N  \l__tbl_tabletag_tl
13 \tl_set:Nn \l__tbl_tabletag_tl {\UseStructureName{table}}
```

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 {}
```

Variable to store cell info like which cell must be skipped when tagging multirows.

```
\prop_new:N\g__tbl_untagged_cells_prop
\prop_new:N\l__tbl_saved_untagged_cells_prop
```

Temp variables

```
\clist_new:N \l__tbl_tmpa_clist
\tl_new:N \l__tbl_tmpa_tl
\str_new:N \l__tbl_tmpa_str
```

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

5.2 Tagging support sockets

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

The `tbl/init/celldata` and `tbl/restore/celldata` are defined in lttagging.

`default` (*plug*) This socket is used inside `\tbl_init_cell_data_for_table` for the case that this a nested table in a cell.

```
\NewTaggingSocketPlug{tbl/init/celldata}{default}
  {
    \prop_set_eq:NN\l__tbl_saved_untagged_cells_prop \g__tbl_untagged_cells_prop
    \prop_gclear:N \g__tbl_untagged_cells_prop
  }
\AssignTaggingSocketPlug{tbl/init/celldata}{default}
```

`default` (*plug*) This socket is used inside `\tbl_restore_outer_cell_data:` and restores values when a nested table is ended.

```
\NewTaggingSocketPlug{tbl/restore/celldata}{default}
  {
    \prop_gset_eq:NN\g__tbl_untagged_cells_prop \l__tbl_saved_untagged_cells_prop
  }
\AssignTaggingSocketPlug{tbl/restore/celldata}{default}
```

`TD` (*plug*)

```
\NewTaggingSocketPlug{tbl/cell/begin}{TD}
  {
```

Next line was previously outside of the plug, so if we want to execute it always even if the noop plug is in force this needs a different solution.

```
    \__tbl_show_curr_cell_data:
```

We test if the cell should be tagged at all.

```
    \__tbl_if_tag_cell:nnT {\g__tbl_row_int } { \g__tbl_col_int }
      {
```

this sets row headers

```
        \clist_if_in:NVT \l__tbl_header_columns_clist\g__tbl_col_int
          {
            \tl_set:Ne \l__tbl_celltag_tl {\UseStructureName{table/headercell}}
            \tl_set:Ne \l__tbl_cellattribute_tl {\l__tbl_cellattribute_tl,TH-row}
          }
```

Here we handle negative value for row headers:

```
        \tl_set:Ne \l__tbl_tmpa_tl
          {\int_eval:n { \g__tbl_col_int - 1 - \g__tbl_table_cols_tl }}
        \clist_if_in:NoT \l__tbl_header_columns_clist { \l__tbl_tmpa_tl }
          {
            \tl_set:Ne \l__tbl_celltag_tl {\UseStructureName{table/headercell}}
            \tl_set:Ne \l__tbl_cellattribute_tl {\l__tbl_cellattribute_tl,TH-row}
          }
        \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*)

```
\NewTaggingSocketPlug{tbl/cell/end}{TD}
  {
    \__tbl_if_tag_cell:nnT {\g__tbl_row_int } { \g__tbl_col_int }
      {
        \tag_mc_end:
        \tag_struct_end:
      }
  }
```

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

TDpbox (*plug*)

```
\NewTaggingSocketPlug{tbl/pcell/begin}{TDpbox}
  {
    \__tbl_show_curr_cell_data:
    \__tbl_if_tag_cell:nnT {\g__tbl_row_int } { \g__tbl_col_int }
      {
        \clist_if_in:NVT \l__tbl_header_columns_clist\g__tbl_col_int
          {
            \tl_set:Ne \l__tbl_pcelltag_tl {\UseStructureName{table/headercell}}
```

```
            \tl_set:Ne \l__tbl_cellattribute_tl {\l__tbl_cellattribute_tl,TH-row}
          }
        \tag_struct_begin:n
          {
            tag             =\l__tbl_pcelltag_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
        \AssignStructureRole{para/semantic}{\UseStructureName{table/para/semantic}}
     }
  }
```

TDpbox (*plug*)

```
\NewTaggingSocketPlug{tbl/pcell/end}{TDpbox}
  {
    \__tbl_if_tag_cell:nnT {\g__tbl_row_int } { \g__tbl_col_int }
      {
        \tag_struct_end:
        \legacy_if:nT {@endpe}{\par}
        \mode_if_vertical:T{ \tagpdfparaOff }
      }
  }
```

TR (*plug*)

```
\NewTaggingSocketPlug{tbl/row/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*)

```
\NewTaggingSocketPlug{tbl/row/end}{TR}
  {
    \__tbl_add_missing_cells:
    \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 }
      {
        \__tbl_trace:n
          {==>~
            structure~stored~for~row~\int_use:N\g__tbl_row_int :~
```

```
            \seq_use:Nn \g__tbl_struct_cur_seq {,}
          }
      }
      { \ERRORtbl/row } % 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 TD or TR Part is not allowed as child. For structures that restore paragraph settings we therefore need a special plug that adjust the settings. Currently we set the para main tag to Div. This leads to double Div-structures but flattening the paragraph doesn't work, it errors if there is a list inside.

```
\NewTaggingSocketPlug{para/restore}{Table}
  {
    \AssignStructureRole{para/semantic}{\UseStructureName{table/para/semantic}}
    \AssignStructureRole{para/textblock}{\c_tag_para_textblock_tl}
    \bool_set_true:N \l__tag_para_bool
    \bool_set_false:N \l__tag_para_flattened_bool
    \int_zero:N \l__tag_block_flattened_level_int
  }
```

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

```
\NewTaggingSocketPlug{tbl/init}{Table}
  {
     \bool_set_false:N \l__tag_para_bool
     \AssignTaggingSocketPlug{para/restore}{Table}
```

We also initialize the structure data variables a this point.

```
     \__tbl_init_struct_data:
  }
```

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

```
\NewTaggingSocketPlug{tbl/finalize}{Table}
  {
    \__tbl_set_header_rows:
```

Similarly, we restore the outer values of the structure data when we leave the table.

```
    \__tbl_restore_struct_data:
  }
```

`Table` (*plug*) This plug will initialize the structure in longtable.

```
\NewTaggingSocketPlug{tbl/longtable/init}{Table}
 {
   \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
 }
```

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

```
\NewTaggingSocketPlug{tbl/longtable/finalize}{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 }
                      {
                        \tag_struct_gput:ene { \l__tbl_tmpa_tl }{tag}{/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 }
                      {
                        \tag_struct_gput:ene { \l__tbl_tmpa_tl }{tag}{/Artifact}
                      }
                  }
              }
          }
      }
  }
```

`Table` (*plug*)

We must avoid that the reuse of the header foot box leads to duplicated content, thus reset attribute of the box:

```
\NewTaggingSocketPlug{tbl/longtable/head}{Table}
 {
  \tagmcbegin{artifact}
   \tag_mc_reset_box:N\LT@head
  \tagmcend
 }
```

Table (*plug*)

```
\NewTaggingSocketPlug{tbl/longtable/foot}{Table}
 {
   \tagmcbegin{artifact}
   \tag_mc_reset_box:N \LT@foot
   \tagmcend
 }
```

Table (*plug*)

```
\NewTaggingSocketPlug{tbl/hmode/begin}{Table}
  {
    \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.

```
     \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=\l__tbl_tabletag_tl}
    \tl_gset:Ne \g__tbl_struct_table_tl { \tag_get:n {struct_num} }
  }
```

LayoutTable (*plug*) This plug is meant for presentation tables, it sets the ARIA role.

```
\NewTaggingSocketPlug{tbl/hmode/begin}{LayoutTable}
  {
    \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.

```
     \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=\l__tbl_tabletag_tl,attribute-class=ARIA-role-presentation}
    \tl_gset:Ne \g__tbl_struct_table_tl { \tag_get:n {struct_num} }
  }
```

Table (*plug*)

```
\NewTaggingSocketPlug{tbl/hmode/end}{Table}
  {
    \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.

```
    \bool_lazy_and:nnT
      { \bool_if_exist_p:N \l__tag_para_bool } { \l__tag_para_bool }
      { \tag_struct_begin:n { tag=\UseStructureName{para/textblock} } }
    \tag_mc_begin_pop:n{}
  }
```

Table (*plug*)

```
\NewTaggingSocketPlug{tbl/vmode/begin}{Table}
  {
    \tag_struct_begin:n {tag=\l__tbl_tabletag_tl}
    \tl_gset:Ne \g__tbl_struct_table_tl { \tag_get:n {struct_num} }
  }
```

Table (*plug*)

```
\NewTaggingSocketPlug{tbl/vmode/begin}{LayoutTable}
  {
    \tag_struct_begin:n {tag=\l__tbl_tabletag_tl,attribute-class=ARIA-role-presentation}
    \tl_gset:Ne \g__tbl_struct_table_tl { \tag_get:n {struct_num} }
  }
```

Table (*plug*)

```
\NewTaggingSocketPlug{tbl/vmode/end}{Table}
  {
    \tag_struct_end:
    \par
  }
```

code (*plug*) This socket takes a number, checks if is larger than one, checks if the colspan attribute already exists (we can't predefine an arbitrary number), and updates `\l__tbl_cellattribute_tl`.

```
\NewTaggingSocketPlug{tbl/colspan}{code}
  {
    \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
          {\l__tbl_cellattribute_tl,colspan-\int_eval:n{#1}}
     }
  }
```

code (*plug*) The sockets are declared in lttagging.dtx.

```
\NewTaggingSocketPlug{tbl/leaders/begin}{code}
  { \tag_mc_begin:n{artifact} }
\NewTaggingSocketPlug{tbl/leaders/end}{code}
  { \tag_mc_end: }
```

5.3 Environments

Currently we support only `tabular`, `tabular*`, `tabularx` and `longtable` (and possibly environments build directly on top of them).

The `array` environment is math. So we disable table tagging for now. We use the command hook to catch also cases where `\array` is used directly in other environments like matrix environments. Perhaps table tagging should be disable for math generally, but then we have to handle text insertions.

```
\AddToHook{cmd/array/before}{\__tag_tbl_disable:}
```

5.4 Interfaces to tagging

5.4.1 Tagging helper commands

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:
 {
   \AssignTaggingSocketPlug{tbl/cell/begin}{noop}
   \AssignTaggingSocketPlug{tbl/cell/end}{noop}
   \AssignTaggingSocketPlug{tbl/pcell/begin}{noop}
   \AssignTaggingSocketPlug{tbl/pcell/end}{noop}
   \AssignTaggingSocketPlug{tbl/row/begin}{noop}
   \AssignTaggingSocketPlug{tbl/row/end}{noop}
   \AssignTaggingSocketPlug{tbl/init}{noop}
   \AssignTaggingSocketPlug{tbl/finalize}{noop}
   \AssignTaggingSocketPlug{tbl/longtable/init}{noop}
   \AssignTaggingSocketPlug{tbl/longtable/head}{noop}
   \AssignTaggingSocketPlug{tbl/longtable/foot}{noop}
   \AssignTaggingSocketPlug{tbl/longtable/finalize}{noop}
   \AssignTaggingSocketPlug{tbl/hmode/begin}{noop}
   \AssignTaggingSocketPlug{tbl/hmode/end}{noop}
   \AssignTaggingSocketPlug{tbl/vmode/begin}{noop}
   \AssignTaggingSocketPlug{tbl/vmode/end}{noop}
   \AssignTaggingSocketPlug{tbl/colspan}{noop}
   \AssignTaggingSocketPlug{tbl/leaders/begin}{noop}
   \AssignTaggingSocketPlug{tbl/leaders/end}{noop}
 }
```

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

__tag_tbl_enable:

```
\cs_new_protected:Npn \__tag_tbl_enable:
 {
   \AssignTaggingSocketPlug{tbl/cell/begin}{TD}
   \AssignTaggingSocketPlug{tbl/cell/end}{TD}
   \AssignTaggingSocketPlug{tbl/pcell/begin}{TDpbox}
   \AssignTaggingSocketPlug{tbl/pcell/end}{TDpbox}
   \AssignTaggingSocketPlug{tbl/row/begin}{TR}
   \AssignTaggingSocketPlug{tbl/row/end}{TR}
   \AssignTaggingSocketPlug{tbl/init}{Table}
   \AssignTaggingSocketPlug{tbl/finalize}{Table}
   \AssignTaggingSocketPlug{tbl/longtable/head}{Table}
   \AssignTaggingSocketPlug{tbl/longtable/foot}{Table}
   \AssignTaggingSocketPlug{tbl/longtable/init}{Table}
   \AssignTaggingSocketPlug{tbl/longtable/finalize}{Table}
   \AssignTaggingSocketPlug{tbl/hmode/begin}{Table}
   \AssignTaggingSocketPlug{tbl/hmode/end}{Table}
   \AssignTaggingSocketPlug{tbl/vmode/begin}{Table}
   \AssignTaggingSocketPlug{tbl/vmode/end}{Table}
   \AssignTaggingSocketPlug{tbl/colspan}{code}
```

```
    \AssignTaggingSocketPlug{tbl/leaders/begin}{code}
    \AssignTaggingSocketPlug{tbl/leaders/end}{code}
  }
```

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

`\__tbl_init_tagnames_default:`
`\__tbl_init_tagnames_div:`

These commands are used to switch the tagnames.

```
\cs_new_protected:Npn\__tbl_init_tagnames_default:
  {
    \tl_set:Nn\l__tbl_rowtag_tl   {\UseStructureName{table/row}}
    \tl_set:Nn\l__tbl_pcelltag_tl {\UseStructureName{table/cell}}
    \tl_set:Nn\l__tbl_celltag_tl  {\UseStructureName{table/cell}}
    \tl_set:Nn\l__tbl_tabletag_tl {\UseStructureName{table}}
  }

\cs_new_protected:Npn\__tbl_init_tagnames_div:
  {
    \tl_set:Nn\l__tbl_rowtag_tl   {\UseStructureName{layouttable/row}}
    \tl_set:Nn\l__tbl_pcelltag_tl {\UseStructureName{layouttable/pcell}}
    \tl_set:Nn\l__tbl_celltag_tl  {\UseStructureName{layouttable/cell}}
    \tl_set:Nn\l__tbl_tabletag_tl {\UseStructureName{layouttable}}
  }
```

(*End of definition for* `\__tbl_init_tagnames_default:` *and* `\__tbl_init_tagnames_div:`.)

```
% See tagpdfsetup-keys.md in tagpdf/doc for the naming scheme.
\keys_define:nn { __tag / setup }
  {
    table/tagging .choices:nn = { true, on }
      {
        \__tag_tbl_enable:
        \__tbl_init_tagnames_default:
      },
    table/tagging .choices:nn = { false, off }
      { \__tag_tbl_disable: },
    table/tagging .choice:,
    table/tagging / presentation .code:n =
      {
        \__tag_tbl_enable:
        \__tbl_init_tagnames_default:
        \AssignTaggingSocketPlug{tbl/hmode/begin}{LayoutTable}
        \AssignTaggingSocketPlug{tbl/vmode/begin}{LayoutTable}
        \clist_clear:N \l__tbl_header_rows_clist
        \clist_clear:N \l__tbl_header_columns_clist
      },
    table/tagging / div .code:n =
      {
        \__tag_tbl_enable:
        \__tbl_init_tagnames_div:
        \AssignTaggingSocketPlug{tbl/hmode/begin}{LayoutTable}
        \AssignTaggingSocketPlug{tbl/vmode/begin}{LayoutTable}
        \clist_clear:N \l__tbl_header_rows_clist
        \clist_clear:N \l__tbl_header_columns_clist
      },
    table/tagging .default:n = true,
```

```
    table/tagging .initial:n = true
  }
```

This are the old key names kept for now for compatibility. They will got at some time.

```
\keys_define:nn { __tag / setup }
  {
    table-tagging .meta:n = {table/tagging={#1}}
  }
```

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.

The attributes `TH-col`, `TH-row` and `TH-both` (for the scope of a header column) and `ARIA-role-presentation` are declared in latex-lab-namespace.

`\l__tbl_header_rows_clist`
`\l__tbl_header_columns_clist`

This holds the numbers of the header rows and columns. Negative numbers are possible and count from the last column backwards.

```
\clist_new:N  \l__tbl_header_rows_clist
\clist_new:N  \l__tbl_header_columns_clist
```

`\__tbl_set_header_rows:`

```
\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.

```
            \tag_if_struct_exist:nT { ####1 }
              {
                \tag_struct_gput:nne{####1}{tag}{/\UseStructureName{table/headercell}}
```

This sets the scope class. If the header has already a row attribute we replace by TH-both.

```
                \tag_get:nnN {####1}{struct_C}\l__tbl_tmpa_tl
                \tl_if_empty:NTF \l__tbl_tmpa_tl
                 {
                   \tag_struct_gput:nnn{####1}{C}{/TH-col}
                 }
                 {
                   \str_set:Ne \l__tbl_tmpa_str {\l__tbl_tmpa_tl}
                   \str_if_in:NnTF\l__tbl_tmpa_str{/TH-row}
                     {
                       \str_replace_once:Nnn \l__tbl_tmpa_str {/TH-row}{/TH-both}
                       \tag_struct_gput:nne{####1}{C}{\l__tbl_tmpa_str}
                     }
                     {
                       \tag_struct_gput:nne{####1}{C}{/TH-col,\l__tbl_tmpa_str}
                     }
                 }
             }
          }
       }
  }
```

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

And some key support: (See tagpdfsetup-keys.md for the naming scheme.)

```
\keys_define:nn { __tag / setup }
  {
    table/header-rows .clist_set:N = \l__tbl_header_rows_clist,
    table/header-columns .clist_set:N = \l__tbl_header_columns_clist,
```

obsolete older name:

```
    table-header-rows .meta:n = {table/header-rows={#1}}
  }
```

5.5 Multirow support

`\__tbl_multirow:n` This command makes the current cell into a multirow cell: it creates, if needed, an RowSpan-attribute, adds it to the attributes of the cell structure, and marks all following cells spanned by the multirow as cells that should not be tagged. The argument is the number of spanned row (including the current row).

```
\cs_if_exist:NT \__tag_struct_prop_gput:nnn
 {
   \cs_generate_variant:Nn \__tag_struct_prop_gput:nnn {one}
 }
\cs_new_protected:Npn \__tbl_multirow:n #1
 {
```

Create an attribute if needed:

```
   \prop_get:NeNF \g__tag_attr_entries_prop
      {rowspan-\int_eval:n{#1}}
      \l__tbl_tmpa_tl
      {
        \__tag_attr_new_entry:ee
```

```
            {rowspan-\int_eval:n{#1}}
            {/O /Table /RowSpan~\int_eval:n{#1}}
        }
```

ensure that the attribute is marked as used:

```
    \seq_gput_left:Ne\g__tag_attr_class_used_seq
      {\pdf_name_from_unicode_e:n{rowspan-\int_eval:n{#1}}}
```

Get the structure number of the current cell

```
    \seq_get_right:NN\g__tbl_struct_cur_seq \l__tbl_tmpb_tl
```

If we are in a multicolumn the number can be negative and this must be changed

```
    \tl_set:Ne \l__tbl_tmpb_tl { \int_abs:n{\l__tbl_tmpb_tl} }
```

no we must update an existing attribute. TODO: simplify this ... (see also colspan handling).

```
    \prop_get:cnNTF
       { g__tag_struct_\l__tbl_tmpb_tl _prop }
       { C }
       \l__tbl_tmpa_tl
       {
         \tl_remove_once:Nn \l__tbl_tmpa_tl {[}
         \tl_remove_once:Nn \l__tbl_tmpa_tl {]}
         \__tag_struct_prop_gput:one{ \l__tbl_tmpb_tl }
           {C}
           {[/rowspan-\int_eval:n{#1}~\l__tbl_tmpa_tl]}
       }
       {
         \__tag_struct_prop_gput:one{ \l__tbl_tmpb_tl }
           {C}
           {[/rowspan-\int_eval:n{#1}]}
       }
```

Now mark the spanned cells that should be ignored.

```
     \__tbl_gset_untagged_row_cells:nn {#1-1}{\g__tbl_span_tl}
  }
```

(*End of definition for* `\__tbl_multirow:n`.)

```
\keys_define:nn{ __tag / setup }
 { table/multirow .code:n = {\__tbl_multirow:n {#1} }
  ,table/multirow .default:n = 1
 }
```

`\__tbl_gset_untagged_row_cells:nn` This command stores the row and column numbers of the cells in the following row(s) that should not be tagged as they are a part of a rowspan.

```
\cs_new_protected:Npn \__tbl_gset_untagged_row_cells:nn #1 #2
  % #1 number of rows, #2 number of columns
  {
    \int_step_inline:nn {#1}
     {
       \int_step_inline:nn {#2}
        {
          \prop_gput:Nee \g__tbl_untagged_cells_prop
           {
```

```
              \int_eval:n {\g__tbl_row_int + ##1},
              \int_eval:n{\g__tbl_col_int + ####1 -1 }
            }{}
        }
    }
}
```

(*End of definition for* `\__tbl_gset_untagged_row_cells:nn`.)

`\__tbl_if_tag_cell:nn` We must be able to detect if a cell should be tagged. For this we define a conditional — if more options are needed it can be extended.

```
\prg_new_protected_conditional:Npnn\__tbl_if_tag_cell:nn #1 #2 %#1 row, #2 col
  { T,TF }
  {
    \prop_get:NeNTF \g__tbl_untagged_cells_prop
      {\int_eval:n{#1},\int_eval:n{#2}}\l__tbl_tmpa_tl
      { \prg_return_false:}
      { \prg_return_true: }
  }
```

(*End of definition for* `\__tbl_if_tag_cell:nn`.)

5.6 Misc stuff

`\__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: {
   \__tbl_trace:n { ==>~ 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:`.)

`\__tbl_add_missing_cells:` 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 one adding is used in the row/end socket.

```
\cs_new:Npn \__tbl_add_missing_cells:
  {
```

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 \g__tbl_missing_cells_int > 0
      {
       \__tbl_trace:n {==>~
         ~Inserting~\int_use:N \g__tbl_missing_cells_int \space
          additional~cell(s)~into~previous~row:}
       \int_step_inline:nn { \g__tbl_missing_cells_int }
           {
             \int_gincr:N\g__tbl_col_int
             \UseTaggingSocket{tbl/cell/begin}
             \UseTaggingSocket{tbl/cell/end}
           }
      }
  }
```

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

`\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.*)

`\__tbl_init_struct_data:`

Save the global structure data variables locally so the we can restore them when the table ends (important in case of nested tables).
This is also the right point to initialize them. `\g__tbl_struct_table_tl` is set elsewhere.

```
\cs_new_protected:Npn \__tbl_init_struct_data: {
  \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
%
  \seq_gclear:N\g__tbl_struct_rows_seq
  \seq_gclear:N\g__tbl_struct_cells_seq
  \seq_gclear:N\g__tbl_struct_cur_seq
}
```

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

`\__tbl_restore_struct_data:`

```
\cs_new_protected:Npn \__tbl_restore_struct_data: {
  \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
}
```

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

5.7 longtable

`\__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{%
```

```
    \AssignTaggingSocketPlug{parbox/before}{noop}
    \AssignTaggingSocketPlug{parbox/after}{noop}
    \hbox to\z@{\hss\parbox[t]\LTcapwidth{%
    \reset@font
    \tag_suspend:n{caption}
    \sbox\@tempboxa{#1{#2:~}#3}%
    \tag_resume: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}
  {
   \cs_set_eq:NN \LT@makecaption\__tbl_patch_LT@makecaption
  }
```

⟨/package⟩