

# The `arydshln` package\*

Hiroshi Nakashima  
(Kyoto University)

2018/09/26

## Abstract

This file gives L<sup>A</sup>T<sub>E</sub>X's `array` and `tabular` environments the capability to draw horizontal/vertical dash-lines.

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Usage</b>	<b>3</b>
2.1	Loading Package . . . . .	3
2.2	Basic Usage . . . . .	4
2.3	Style Parameters . . . . .	4
2.4	Fine Tuning . . . . .	5
2.5	Finer Tuning . . . . .	5
2.6	Performance Tuning . . . . .	6
2.7	Compatibility with Other Packages . . . . .	7
<b>3</b>	<b>Known Problems</b>	<b>9</b>
<b>4</b>	<b>Implementation</b>	<b>10</b>
4.1	Problems and Solutions . . . . .	10
4.2	Another Old Problem . . . . .	13
4.3	Register Declaration . . . . .	14
4.4	Initialization . . . . .	17
4.5	Making Preamble . . . . .	21
4.6	Building Columns . . . . .	27
4.7	Multi-columns . . . . .	29
4.8	End of Rows . . . . .	31
4.9	Horizontal Lines . . . . .	33
4.10	End of Environment . . . . .	36
4.11	Drawing Vertical Lines . . . . .	38

---

\*This file has version number v1.74, last revised 2018/09/26.

4.12	Drawing Dash-lines . . . . .	43
4.13	Shorthand Activation . . . . .	45
4.14	Compatibility with <code>colortab</code> . . . . .	47
4.15	Compatibility with <code>longtable</code> . . . . .	48
4.15.1	Initialization . . . . .	48
4.15.2	Ending Chunks . . . . .	50
4.15.3	Horizontal Lines and <code>p</code> -Boxes . . . . .	52
4.15.4	First Chunk . . . . .	54
4.15.5	Output Routine . . . . .	55
4.16	Compatibility with <code>colortbl</code> . . . . .	60
4.16.1	Initialization, Cell Coloring and Finalization . . . . .	61
4.16.2	Horizontal Line Coloring . . . . .	63
4.16.3	Vertical Line Coloring . . . . .	64
4.16.4	Compatibility with <code>longtable</code> . . . . .	67

# 1 Introduction

In January 1993, Weimin Zhang kindly posted a style `hvdashln` written by the author, which draws horizontal/vertical dash-lines in L<sup>A</sup>T<sub>E</sub>X's `array` and `tabular` environments, to the news group `comp.text.tex`. The style, unfortunately, has a known problem that vertical lines are broken when an array contains tall rows.

In March of the year, Monty Hayes complained of this problem encouraging the author to make a new version `arydshln` to solve the problem. The new style also has new features, such as allowing ‘:’ to specify a vertical dash-line in preamble, and `\cdashline` being a counterpart of `\cline`.

In March 1999, Sebastian Rahtz kindly invited the style, which had been improved following the bug report from Takahiro Kubota, to be included in T<sub>E</sub>X CTAN and also in the online catalogue compiled by Graham Williams. This invitation gave the style new users including Peter Ehrbar who wished to use it with `array` style in Standard L<sup>A</sup>T<sub>E</sub>X Tools Bundle and had trouble because these styles were incompatible with each other. Therefore, the style became compatible with `array` and got additional new features.

In February 2000, Zsuzsanna Nagy reported that `arydshln` is not compatible with `colortab` style to let the author work on the compatibility issue again.

In February 2001, Craig Leech reported another compatibility problem with `longtable`. Although the author promised that the problem would be attacked some day, the issue had left long time<sup>1</sup> until three other complaints were made. Then the author attacked the problem hoping it is the last compatibility issue<sup>2</sup>.

In May 2004, Klaus Dalinghaus found another incompatibility with `colortbl`. Although he was satisfied by a quick hack for cell painting, the author attacked a harder problem for line coloring to solve the problem<sup>3</sup>.

## 2 Usage

### 2.1 Loading Package

The package is usable to both L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> and L<sup>A</sup>T<sub>E</sub>X-2.09 users with their standard package loading declaration. If you use L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>, simply do the following.

```
\usepackage{arydshln}
```

If you still love L<sup>A</sup>T<sub>E</sub>X-2.09, the following is what you have to do.

```
\documentstyle[...arydshln,...]{<style>}
```

Only one caution given to users of `array` (v2.3m or later) and `longtable` (v4.10 or later) packages, included in Standard L<sup>A</sup>T<sub>E</sub>X Tools Bundle, and `colortab` and `colortbl` package is that `arydshln` has to be loaded *after* `array`, `longtable`, `colortab` and/or `colortbl` done. That is, the following is correct but reversing the order of `\usepackage` will cause some mysterious error.

---

<sup>1</sup>Two years and a half! Sorry Craig.

<sup>2</sup>But his hope was dashed as described below.

<sup>3</sup>Without dreaming it is the last compatibility issue.

```

\usepackage{array}      % and/or
\usepackage{longtable} % and/or
\usepackage{colortab}   % or
\usepackage{colortbl}
\usepackage{arydshln}

```

## 2.2 Basic Usage

`array` You can simply use `array` or `tabular(*)` environments with standard preamble, such as `tabular {r|c|ll}`, and standard commands `\\`, `\hline`, `\cline` and `\multicolumn`.  
`:` Drawing a vertical dash-line is quite simple. Use ‘:’ in the preamble as the separator of columns separated by the dash-line, just like using ‘|’ to draw a vertical solid-line. The *preamble* means not only that of the environment, but also the first argument of `\multicolumn`.

`\hdashline` It is also simple to draw a horizontal dash-line. Use `\hdashline` and `\cdashline` as the counterparts of `\hline` and `\cline`.  
`\cdashline`

For example;

```

\begin{tabular}{|l::c:r|}\hline
A&B&C\\hdashline
AAA&BBB&CCC\\cdashline{1-2}
\multicolumn{2}{|l:}{AB}&C\\hdashline\hdashline
\end{tabular}

```

will produce the following result.

A	B	C
AAA	BBB	CCC
AB		C

Note that the intersections of leftmost/rightmost vertical lines and horizontal dash-lines are little bit different from those produced by ordinary `array/tabular`. That is, with very careful examination you will find that vertical lines of ordinary ones are *broken* with small white specks at intersections, while in the example above they have no specks. In addition, the four corners of outermost rectangular also have specks in ordinary ones, while those in the example above have perfect contacts of L-shape<sup>4</sup>.

`\firsthdashline` If you use `array`, the dashed version of `\firsthline` and `\lasthline` named `\firsthdashline` and `\lasthdashline` are available.

## 2.3 Style Parameters

`\dashlinedash` You have two style parameters to control the shape of dash-lines: `\dashlinedash` is for the length of each dash segment in a dash line; `\dashlinegap` controls the amount of each gap between dash segments. Both parameters have a common default value, 4 pt.

---

<sup>4</sup>The top-left/right corners had specks before v1.73, the fix in which made the topmost dash segment of a vertical dash-line a little bit shorter.

## 2.4 Fine Tuning

- ; Although you can control the shape of dash-lines in an `array`/`tabular` environment as described in §2.3, you might want to draw a dash-line of a shape different from others. To specify the shape of a vertical dash-line explicitly, you may use;

`;\langle dash\rangle/\langle gap\rangle\}`

instead of ordinary ‘:’ and will have a dash-line with dash segments of `\langle dash\rangle` long separated by spaces of `\langle gap\rangle`.

`\hdashline` As for horizontal dash-lines, explicit shape specifications may be given through optional arguments of `\hdashline` and `\cdashline` as follows.

`\hdashline[\langle dash\rangle/\langle gap\rangle]`  
`\cdashline{\langle col1\rangle-\langle col2\rangle}[\langle dash\rangle/\langle gap\rangle]`

For example;

```
\begin{tabular}{|l:c;{2pt/2pt}r|}\hline
A&B&C\\hdashline[1pt/1pt]
AAA&BBB&CCC\\cdashline{1-2}[.4pt/1pt]
\multicolumn{2}{|l;{2pt/2pt}}{AB}&C\\hdashline\hdashline
\end{tabular}
```

will produce the following result.

A	B	C
AAA	BBB	CCC
AB		C

`\ADLnullwide` The vertical solid and dashed lines are drawn as if their width is zero, as standard  
`\ADLsomewide` L<sup>A</sup>T<sub>E</sub>X’s `array` and `tabular` do, if you don’t use `array` package. Otherwise, they have *real* width of `\arrayrulewidth` as the authors of `array` prefers. However, you may explicitly tell `arydshln` to follow your own preference by `\ADLnullwide` if you love L<sup>A</sup>T<sub>E</sub>X standard, or `\ADLsomewide` if you second the preference of `array` authors.

## 2.5 Finer Tuning

To draw dash-lines, we use a powerful primitive of T<sub>E</sub>X called `\xleaders`. It replicates a segment that consist of a dash and gap so that a dash-line has as many segments as possible and distributes *remainder* space to make the spaces between adjacent dash segments (almost) equal to each other. Therefore, you will have dash-lines with consistent steps of gaps and spaces the lines in Figure 1(1) are.

However, because of a bug (or buggy feature) of `\xleaders`, there *had been* a small possibility that a dash segment near the right/bottom end drops, until it was fixed in the version of 3.141592<sup>5</sup>. Though the fix ultimately made any effort to cope with the problem unnecessary, the package still gives you alternative *drawing modes* which you may specify by `\ADLdrawingmode{\langle m\rangle}` as follows.

A	A	A
B	B	B
C	C	C

(1)

A	A	A
B	B	B
C	C	C

(2)

A	A	A
B	B	B
C	C	C

(3)

Figure 1: Drawing mode controlled by `\ADLdrawingmode`

`\ADLdrawingmode`

- $m = 1$

As shown in Figure 1(1), it gives most beautiful result by `\xleaders`<sup>6</sup>. This is default.

- $m = 2$

As shown in (2) of the figure, beautiful if dash-lines are not so sparse as right/lower lines, but dash segments near the both ends may be a little bit too long as left/upper lines, because in this mode the second first/last segments are drawn by a special mechanism.

- $m = 3$

As shown in (3) of the figure, beautiful if dash-lines are not so sparse as right/lower lines, but gaps near the both ends may be considerably too large as left/upper lines, because in this mode the lines are drawn by `\cleaders`.

It is strongly recommended to use default mode 1 unless you want to have some special effect.

## 2.6 Performance Tuning

Since drawing dash-lines is a hard job, you have to be patient with the fact that the performance of typesetting `array/tabular` with dash-lines is poorer than that of ordinary ones. In fact, according to author's small performance evaluation with a `tabular` having nine vertical and ten horizontal dash-lines, typesetting the `tabular` is approximately ten times as slow as its ordinary counterpart with solid lines.

However, this is not a really bad news, unfortunately. The real one is that loading `arydshln` makes typesetting `array/tabular` slower even if they only have solid lines which the package treats as special ones of dash-lines. The evaluation result shows the degradation factor is about nine. Therefore, if your document has many `array/tabular` with solid lines,  $\text{\LaTeX}$  will run slowly even with quite few (or no) `array/tabular` with dash-lines,

`\ADLinactivate`

To cope with this problem, you may inactivate dash-line functions by the command `\ADLinactivate` that replaces dash-lines with solid lines drawn by a faster (i.e. ordinary) mechanism. Although the inactivation does not completely solve the performance problem, the degradation factor will become much smaller and acceptable, approximately 1.5 in

<sup>5</sup>By pointing out this problem, the author got a check of \$327.68 plus a significantly large amount of interest from DEK. Wow!!

<sup>6</sup>Until the fix of `\xleaders`, the second bottom/rightmost segments of right/lower lines were dropped.

the author's evaluation. For example, the draft version of your document will have the command in its preamble, which you will remove from your final version.

`\ADLactivate` Alternatively, you may do `\ADLinactivate` in the preamble, switch on by `\ADLactivate` before you really need dash-lines, and switch off again afterword. A wiser way could be surrounding `array/tabular` by `\begin{ADLactivate}` and `\end{ADLactivate}`.

`Array` If you feel it tiresome to type the long command/environment name for the activation, `Tabular` you may use `Array` and `Tabular(*)` environment in which dash-line functions are always active. Note that, however, since these environment names are too natural to keep them from being used by authors of other packages or yourself, name conflict could occur. If `Array` and/or `Tabular` have already been defined when `arydshln` is loaded, you will get a warning to show you have to define new environments, say `dlarray` and `dltabular`, as follows.

```
\newenvironment{dlarray}{\ADLactivate\begin{array}}%
{\end{array}}
\newenvironment{dltabular}{\ADLactivate\begin{tabular}}%
{\end{tabular}}
\newenvironment{dltabular*}{\ADLactivate\begin{tabular*}}%
{\end{tabular*}}
```

`\ADLnoshorthanded` On the other hand, if they are defined after `arydshln` is loaded, their definitions are silently replaced or  $\text{\LaTeX}$  complains of multiple definitions. The error in the latter case will be avoided by putting `\ADLnoshorthanded` just after `\usepackage{arydshln}`.

## 2.7 Compatibility with Other Packages

Users of `array` package may use all of newly introduced preamble characters, such as `'>`, `'<`, `'m`, `'b`, and all the commands such as `\extrarowheight`, `\firsthline` and `\lasthline`. The preamble characters given by `arydshln` may be included in the second argument of `\newcolumntype`.

Also users of `colortab` package may use `\LCC/\ECC` construct to color columns. A horizontal solid/dash line may be colored by, e.g. `\NAC\hdashline\ENAC`. The pair of `\AC` and `\EAC` may be used to color everything between them *but*, unfortunately, vertical lines are not. There are no ways to color vertical lines in a table having dash lines. You may color vertical lines of an ordinary table inactivating dash line functions by `\ADLinactivate`.

Another (and more convenient) table coloring tool `colortbl` may be also used simply by loading it before `arydshln`. Not only the painting commands `\rowcolor`, `\columncolor` and `\cellcolor` work well, but both solid and dash lines are also colored by the command `\arrayrulecolor` of `colortbl`<sup>7</sup>. One caution is that `\arrayrulecolor` defines the color of the dash-part of dash lines and thus gap-part has no color (i.e. color of the paper on which the line drawn). Therefore, if you have a `tabular` like;

```
\begin{tabular}{|>\columncolor{red}}1:>\columncolor{green}}r|}
...
\end{tabular}
```

<sup>7</sup>The `colortbl` manual says `\arrayrulecolor` and `\doublerulesepcolor` may be in `>\{...\}` in a preamble but they cause an error with the original implementation. This bug is fixed in `arydshln` and they are now usable to specify the color of the vertical (dash) lines whose specifications occur after the commands.

you will find the vertical dash line is a sequence of black (or the color of `\arrayrulecolor`) and white segments. This problem is partly solved by declaring `\ADLnullwide`<sup>8</sup> to conjunct the red and blue columns and to draw the dash line on their border.

`\ADLnullwidehline`  
`\ADLsomewidehline`

Unfortunately, however, `\ADLnullwide` does not affect the real width of horizontal (dash) lines and thus you will still see white gaps in `\hdashline` and `\cdashline`. A solution is to put `\ADLnullwidehline` before you start a `array/tabular`<sup>9</sup>. With this command, a horizontal (dash) line is drawn adjusting its bottom edge to that of the row above. The command `\ADLsomewidehline` turns the switch to default and the top edge of a horizontal (dash) line will be adjusted to the bottom edge of the row above.

`\dashgapcolor`  
`\nodashgapcolor`

Another method to avoid white gaps is to give a color to gaps by `\dashgapcolor` with arguments same as `\color`. For example;

`\arrayrulecolor{green}\dashgapcolor[rgb]{1,1,0}`

makes colorful dash lines with green dashes and yellow gaps. The command can be placed outside of `array/tabular` for dash lines in the environment, in the argument of preamble character `>` for vertical dash lines following them, or at the beginning of a row for horizontal dash lines following the command. The command `\nodashgapcolor` (no arguments) nullifies the effect of `\dashgapcolor`. Note that `\nodashgapcolor` is different from `\dashgapcolor{white}` because the former makes gaps *transparent* while the later whiten them.

`longtable`  
`Longtable`

Usage of `longtable` with `arydshln` is quite simple. Just loading `arydshln` after `longtable` is enough to make the `longtable` environment able to draw dash-lines. A shorthand activation of dash-line functions is also available by `Longtable` environment. One caution to `longtable` users is that the temporary results before the *convergence* of the column widths may be different from those without `arydshln`. For example, the following is the first pass result of the example shown in Table 3 of the `longtable` manual.

1	2	3
wide multicolumn spanning 1-3		
multicolumn 1-2		3
wide 1	2	3

Since `LTchunksize` is one in the example, columns of each row has their own widths and thus has vertical lines drawn at the edges of the columns. On the other hand, you will have the following as the first pass result with `arydshln`.

1	2	3
wide multicolumn spanning 1-3		
multicolumn 1-2		3
wide 1	2	3

As you see, the vertical lines are drawn at the column edges of the last row<sup>10</sup> because `arydshln` draws them when it see the last row. Anyway, you may ignore temporary results and will have a compatible result when the column widths are converged like the following.

<sup>8</sup>Since `colortbl` automatically loads `array`, the default is `\ADLsomewide`

<sup>9</sup>This command also makes `\cline` and `\cdashline` visible even if the row below is painted.

<sup>10</sup>More precisely, drawn according to the column widths established by all the chunks preceding page output.



1	2	3
wide multicolumn spanning 1-3		
multicolumn 1-2		3
wide 1	2	3

### 3 Known Problems

There are following known problems.

1. The new preamble specifiers ‘:’ and ‘;{*dash*}/<gap>’ cannot be followed or preceded by ‘@{*text*}’, or you will have an ugly result. More specifically, a specifier to draw a dash-line at the left edge of a column cannot be preceded by ‘@{*text*}’, while that to draw at the right edge cannot be followed by ‘@{*text*}’.
2. If you use `array` package, the restriction of ‘@’ shown above is also applied to ‘!’.
3. In order to make it sure that a dash-line always *touches* its both end, i.e. a dash-line always begins and ends with a dash segment, the amount of a gap will slightly vary depending on the dash-line length.
4. If a dash-line is too short, you will have an ugly result without overfull message. More specifically, in mode 1 or 3, a line will look to protrude beyond its column/row borders if it is shorter than a half of `\dashlinedash`. In mode 2, the minimum length to avoid the protrusion is  $1.5 \times \text{\dashlinedash} + \text{\dashlinegap}$ .
5. As described in §2.6, the processing speed for `array` and `tabular` environment will become slower even if dash-lines are not included.
6. As described in §2.7, `\AC` and `\EAC` pair of `colortab` such as `\AC&\EAC` cannot color the vertical line at `&`. Use `\ADLinactivate` if you want to have a ordinary table with colored vertical lines. Note that you may color vertical lines with `colortbl` package.
7. There should be a number of packages whose own `array`/`tabular` implementations are not compatible with `arydshln`, though the author has made efforts at the compatibility. One of them is `plext` package for Japanese typesetting but it has a style file named `plextarydshln.sty` to solve the compatibility issue. So if you use the functionality of `arydshln` with `plext`, do `\usepackage{plextarydshln}` instead of `\usepackage{arydshln}`.

## 4 Implementation

### 4.1 Problems and Solutions

We have two different problems to solve; how to draw horizontal dash-lines and how to draw vertical dash-lines. The former problem is relatively easy because the technique for drawing `\cline`-s can be used. That is, if we know the number of columns, we can draw a dash-line across the `\multispan`-ed columns by `\xleaders` of dash. Modifying a preamble of `array/tabular` to count the number of columns is not hard. Since `\cdashline` is given beginning and ending columns, its implementation is also easy.

The latter problem, however, is much harder. Remember that `array/tabular` draws vertical solid lines by `\vrule`-s in each row without height/depth specification exploiting  $\text{\TeX}$ 's sophisticated mechanism of the rule extension in the surrounding box. Since  $\text{\TeX}$  does not have such a mechanism for `\xleaders` unfortunately, we at least have to know the height and depth of a row which includes vertical dash-lines. Although the height and depth are often same as those of `\@arstrutbox`, we will have an exceptionally tall and/or deep row that makes dash-lines *broken* if we assume every row has the standard height and depth.

Moreover, even if we can measure the height/depth of each row (in fact we will do as described later), drawing dash-lines in each row will not produce a good result. Look at the following two examples closely.

A	B
A	B

In the left example, two dash-lines are individually drawn in two rows. Since the first row is not so tall and deep (8.4 pt/3.6 pt) as to contain enough number of default dash segments (4 pt dash and 4 pt gap) to keep `\xleaders` from inserting a large space, the dash-line in the first row is *sparse*. On the other hand, the second row is enough tall and deep (16.8 pt/7.2 pt) and thus the dash-line in the row looks better. Thus the resulting dash-line is awful because it does not have a continuous dash/gap sequence.

The right example, which we wish to produce, is much better than the left. In this example, the dash line is drawn across two rows keeping continuous steps of dashes and gaps. In order to have this result, we have to draw the dash-line *after* two rows are built because it is necessary to know the total height and depth of two rows. In general, if we know the total height and depth of rows and whether a column has a dash-line, we can draw dash-lines by adding an extra row containing dash-lines. For example, the result shown above is obtained by the following row.

```
\omit\hss<dash-line of 36 pt high>&\omit\cr
```

Note that `<dash-line of 36 pt high>` have to be `\smash`-ed.

In addition to this basic scheme, we have to take the following points into account.

- A dash-line drawn by the preamble character ‘;’ will have non-default dash/gap specification.

- A column may have two or more dash-lines separated by spaces of `\doublerulesep`. Mixed sequence of solid- and dash-lines also have to be allowed.
- The first column may have dash-lines at both ends, while those of others will appear at right ends only. An exception of this rule is brought by `\multicolumn` that may have leading sequence of solid- and/or dash-line specifiers in its preamble.
- A `\multicolumn` may break or add a dash-line, or may change the dash/gap specification of a dash-line. A sequence of `\h(dash)line`-s also break dash-lines.
- If `colortbl` is in use, coloring dash/gap by `\arrayrulecolor` and `\dashgapcolor` gives another possibility of the variation of dash/gap specification.

In order to cope with them, the following data structure is constructed during rows are built.

1. The list of row information  $R = \langle r_1, r_2, \dots, r_N \rangle$ .
2. The  $i^{th}$  element of  $R$ ,  $r_i$ , is one of the following<sup>11</sup>.
  - (a) A triple  $\langle C_i^L, C_i^R, h_i \rangle$ , where  $C_i^L$  and  $C_i^R$  are the lists of solid- or dash-line segments drawn at the left and right edge of columns respectively, and  $h_i$  is the height plus depth of the  $i^{th}$  row.
  - (b) `connect( $h_i$ )` for a `\h(dash)line` of  $h_i$  wide meaning that  $r_i$  is an empty pseudo row of  $h_i$  high and dash-lines are not broken at the row.
  - (c) In `longtable` environment, `discard( $h_i$ )` for a negative vertical space inserted by `\[\langle h_i \rangle]` or `\h(dash)line` meaning  $r_i$  is an empty pseudo row of  $h_i$  high and dash-lines are not broken but may be discarded by the page break at the row.
  - (d) `disconnect( $h_i$ )` for a vertical gap generated by a sequence of `\h(dash)line` meaning that  $r_i$  is an empty pseudo row of  $h_i$  high and dash-lines are broken at the row.
3.  $C_i^L = \langle e_1^i, e_2^i, \dots, e_m^i \rangle$  where  $e_j^i$  corresponds to the  $j^{th}$  (leftmost is first) solid- or dash-line segment.  $C_i^R$  is similar but its elements are ordered in reverse, i.e. the rightmost segment is the first element.
4. The  $j^{th}$  element of  $C_i^L$  or  $C_i^R$ ,  $e_j^i$ , is a triple  $\langle c_j^i, d_j^i, g_j^i \rangle$  where  $c_j^i$  is the column number in which the segment appears, and  $d_j^i$  and  $g_j^i$  are dash/gap specification, length and color, of the segment. For a solid line segment, the length attributes of both  $d_j^i$  and  $g_j^i$  are 0.

Then this data structure is processed to draw solid- and dash-lines at the end of the `array/tabular` as follows. Let  $e_j^i = \langle c_j^i, d_j^i, g_j^i \rangle$  be the  $j^{th}$  element of  $C_i^L$  of  $r_i$ . The *position*  $p_j^i$  of  $e_j^i$  in the column  $c_j^i$  is defined as follows.

$$p_j^i = \begin{cases} 1 & \text{if } j = 1 \vee c_j^i \neq c_{j-1}^i \\ p_{j-1}^i + 1 & \text{otherwise} \end{cases}.$$

---

<sup>11</sup>In the real implementation, the structure of  $r_i$  is slightly different.

The following defines whether two elements  $e_j^i$  and  $e_{j'}^{i'}$  are *connected*, or  $e_j^i \sim e_{j'}^{i'}$ .

$$\begin{aligned} e_j^i \sim e_{j'}^{i'} &\leftrightarrow i < i' \wedge \\ &c_j^i = c_{j'}^{i'} \wedge d_j^i = d_{j'}^{i'} \wedge g_j^i = g_{j'}^{i'} \wedge p_j^i = p_{j'}^{i'} \wedge \\ &\forall k (i < k < i' \rightarrow r_k \in \{\text{connect}(h_k), \text{discard}(h_k)\}). \end{aligned}$$

With these definitions, we can classify all  $e_j^i$  into ordered sets  $S_1, S_2, \dots, S_n$  as follows.

$$\begin{aligned} k \neq k' &\leftrightarrow S_k \cap S_{k'} = \emptyset \\ e_j^i \sim e_{j'}^{i'} &\leftrightarrow \exists k : e_j^i, e_{j'}^{i'} \in S_k \wedge S_k = \{\dots, e_j^i, e_{j'}^{i'}, \dots\} \\ k < k' &\leftrightarrow \forall e_j^i \in S_k, \forall e_{j'}^{i'} \in S_{k'} : (c_j^i < c_{j'}^{i'}) \vee \\ &\quad (c_j^i = c_{j'}^{i'} \wedge p_j^i < p_{j'}^{i'}) \vee \\ &\quad (c_j^i = c_{j'}^{i'} \wedge p_j^i = p_{j'}^{i'} \wedge i < i'). \end{aligned}$$

Now we can draw a dash-line  $L_k = \langle \gamma_k, \pi_k, \delta_k, \xi_k, \tau_k, \beta_k \rangle$  corresponding to  $S_k = \{e_j^i, \dots, e_{j'}^{i'}\}$  as follows.

- $L_k$  is  $\pi_k^{th}$  line in the  $\gamma_k^{th}$  column where  $\gamma_k = c_j^i = \dots = c_{j'}^{i'}$  and  $\pi_k = p_j^i = \dots = p_{j'}^{i'}$ .
- $L_k$  has the dash specification (size and color)  $\delta_k = d_j^i = \dots = d_{j'}^{i'}$ , and gap specification  $\xi_k = g_j^i = \dots = g_{j'}^{i'}$ .
- The top and bottom ends of  $L_k$  are at  $\tau_k$  and  $\beta_k$  above the bottom of the `array/tabular`, where;

$$\eta_l = \begin{cases} h_l & r_l = \text{connect}(h_l) \\ 0 & \text{otherwise} \end{cases}, \quad \tau_k = \eta_{i-1} + \sum_{l=i}^N h_l, \quad \beta_k = -\eta_{i'+1} + \sum_{l=i'+1}^N h_l.$$

Note that  $\eta_{i-1}$  and  $\eta_{i'+1}$  are added/subtracted so that the top/bottom of  $L_k$  is at the top/bottom edge of the horizontal lines above/below the set  $S_k$ .

The row to draw  $L_1, \dots, L_n$  is;

$$\sigma_1 L_1 \sigma_2 L_2 \dots L_{n-1} \sigma_n L_n \sigma_{n+1} \backslash \text{cr}$$

where;

$$\begin{aligned} \sigma_1 &= \backslash \text{omit}[\backslash \text{hss} \& \backslash \text{omit}]^{\gamma_1-1} \\ \sigma_{1 < k \leq n} &= \begin{cases} \backslash \text{null} & \text{if } \gamma_{k-1} = \gamma_k \wedge \pi_{k-1} = \pi_k \\ \backslash \text{skip} \backslash \text{doublerulesep} & \text{if } \gamma_{k-1} = \gamma_k \wedge \pi_{k-1} \neq \pi_k \\ [\backslash \text{hss} \& \backslash \text{omit}]^{\gamma_k - \gamma_{k-1}} & \text{if } \gamma_{k-1} \neq \gamma_k \end{cases} \\ \sigma_{n+1} &= [\backslash \text{hss} \& \backslash \text{omit}]^{\Gamma - \gamma_n - 1} \backslash \text{hss}. \end{aligned}$$

Note that  $[x]^m$  means  $m$ -times iteration of  $x$ , and  $\Gamma$  is the number of columns specified in the preamble.

Dash-lines at the right edges of columns are similarly drawn by processing  $C_i^R$  with the following modifications.

$$\begin{aligned}
k < k' &\leftrightarrow \forall e_j^i \in S_k, \forall e_{j'}^{i'} \in S_{k'} : (c_j^i < c_{j'}^{i'}) \vee \\
&\quad (c_j^i = c_{j'}^{i'} \wedge p_j^i > p_{j'}^{i'}) \vee \\
&\quad (c_j^i = c_{j'}^{i'} \wedge p_j^i = p_{j'}^{i'} \wedge i < i') \\
\sigma_1 &= \backslashomit\hss[\backslashomit\hss]^{\gamma_1-1} \\
\sigma_{k>1} &= \begin{cases} \backslashnull & \text{if } \gamma_{k-1} = \gamma_k \wedge \pi_{k-1} = \pi_k \\ \backslashhskip\backslashdoublerulesep & \text{if } \gamma_{k-1} = \gamma_k \wedge \pi_{k-1} \neq \pi_k \\ [\backslashomit\hss]^{\gamma_k-\gamma_{k-1}} & \text{if } \gamma_{k-1} \neq \gamma_k \end{cases} \\
\sigma_{n+1} &= [\backslashomit\hss]^{\Gamma-\gamma_n-1}
\end{aligned}$$

## 4.2 Another Old Problem

In the default mode 1, we draw a dash line of dash size  $d$  and gap size  $g$  as follows. Let  $W$  be the length of the line plus  $10\text{sp}$ <sup>12</sup>, which is unknown for us if horizontal but known for  $\text{T}_\text{E}\text{X}$ , and assume  $W \geq d/2$  (or the line protrude to the column/row boarder.) At the both ends of the columns, dashes of  $d/2$  long are drawn to make the dash-line *touched* to the ends. Then  $n = \lfloor (W - d - g)/(d + g) \rfloor$  dashes are equally distributed in the remaining space. Thus we will have;

$$D_0(d/2)G_0(g + \varepsilon')D_1(d)G_1(g + \varepsilon) \dots G_{n-1}(g + \varepsilon)D_n(d)G_n(g + \varepsilon')D_{n+1}(d/2)$$

where  $D_i(l)$  and  $G_i(l)$  are dash and gap of  $l$  long,  $\varepsilon = (W - (n + 1)(d + g))/(n + 1)$  (rounded), and  $\varepsilon' = (W - (n + 1)(d + g) - (n - 1)\varepsilon)/2$  to compensate the rounding error on the calculation of  $\varepsilon$ . For a horizontal line, this result will be obtained by `\xleaders` as follows where  $G_i^m(\varepsilon)$  and  $G_i^m(\varepsilon')$  are the spaces inserted by `\xleaders`.

$$\begin{aligned}
&D_0(d/2)G_0^l(g/2)\backslashxleaders\backslashhbox{\mathstrut}^{G^r(g/2)D(d)G^l(g/2)}\backslashhss G_n^r(g)D_{n+1}(d/2) \\
&= D_0(d/2)G_0^l(g/2)G_0^m(\varepsilon') (G_0^r(g/2)D_1(d)G_1^l(g/2)) G_1^m(\varepsilon) \\
&\quad (G_1^r(g/2)D_2(d)G_2^l(g/2)) G_2^m(\varepsilon) \\
&\quad \dots \\
&\quad G_{n-1}^m(\varepsilon) (G_{n-1}^r(g/2)D_n(d)G_n^l(g/2)) G_n^m(\varepsilon') G_n^r(g/2)D_{n+1}(d/2) \\
&= D_0(d/2)G_0(g + \varepsilon')D_1(d)G_1(g + \varepsilon) \dots G_{n-1}(g + \varepsilon)D_n(d)G_n(g + \varepsilon')D_{n+1}(d/2)
\end{aligned}$$

The problem is that  $\varepsilon'$  could be negative and old  $\text{T}_\text{E}\text{X}$  mistakenly ignored this possibility. That is, since the  $\text{T}_\text{E}\text{X}$  older than 3.141592 did not put `\hbox` beyond the right edge of `\xleaders`, the rightmost `\hbox` was omitted if  $\varepsilon'$  is negative.

Since it is (almost) impossible to know the length of a horizontal line, we could not cope with this problem by adding or subtracting its length. Thus we introduced *drawing mode*

<sup>12</sup>This small amount is added by `\xleaders` in order to, according to the comment in `tex.web`, compensate floating point rounding error.

to have imperfect solutions. In the mode 2, we draw a line by the following sequence.

$$D_0(d/2)G_0^l(g/2)G_{0'}^r(g/2)D_{1'}(d)G_{1'}^l(g/2)G(-d-g) \\ \backslash\mathrm{xleaders}\backslash\mathrm{hbox}\{G^r(g/2)D(d)G^l(g/2)\}\backslash\mathrm{hss} \\ G(-d-g)G_{n'}^r(g/2)D_{n'}(d)G_{n'}^l(g/2)G_n^r(g)D_{n+1}(d/2)$$

That is,  $n^{th}$  `\hbox` that could be disappeared is put twice and the first one is also overlaid for symmetrization. Therefore the length of the first and  $n^{th}$  dashes is  $d + |\varepsilon'|$  and thus could be a little bit longer than others.

On the other hand, we replace `\xleaders` of mode 1 with `\cleaders` for the drawing in mode 3. The result will be;

$$D_0(d/2)G_0(g+R)D_1(d)G_1(g)\dots G_{n-1}(g)D_n(d)G_n(g+R)D_{n+1}(d/2)$$

where  $R = (W - (n+1)(d+g))/2$  to make the first and last gaps considerably wider than others.

### 4.3 Register Declaration

Here registers and switches are declared.

<code>\dashlinedash</code>	First of all, two <code>\dimen</code> registers <code>\dashlinedash</code> and <code>\dashlinegap</code> to control the shape
<code>\dashlinegap</code>	of dash-lines are declared, and their default values, 4pt for both, are assigned to them.
<code>\hdashlinewidth</code>	They have aliases, <code>\hdashlinewidth</code> and <code>\hdashlinegap</code> respectively, for the backward
<code>\hdashlinegap</code>	compatibility.

```

1 %% Register Declaration
2
3 \newdimen\dashlinedash \dashlinedash4pt %
4 \newdimen\dashlinegap \dashlinegap4pt %
5 \let\hdashlinewidth\dashlinedash
6 \let\hdashlinegap\dashlinegap
7

```

Next, the following six switches are declared.

<code>\ifadl@leftrule</code>	• <code>\ifadl@leftrule</code> is used in the preamble analysis macro <code>\@mkpream</code> and is true during it processes leading characters for solid- and dash-lines, i.e. ‘ ’, ‘:’, and ‘;’.
<code>\ifadl@connected</code>	• <code>\ifadl@connected</code> is used to indicate the <i>connection</i> $e_j^i \sim e_{j'}^{i'}$ . When we process $e_{j'}^{i'}$ , the switch is true iff $\exists e_j^i (e_j^i \sim e_{j'}^{i'})$ .
<code>\ifadl@doublerule</code>	• <code>\ifadl@doublerule</code> is used to make $\sigma_k$ . When we are to make $\sigma_k L_k$ , it is true iff $\gamma_{k-1} = \gamma_k \wedge \pi_{k-1} \neq \pi_k$ .
<code>\ifadl@zwvrule</code>	• <code>\ifadl@zwvrule</code> controls the <i>real</i> width of vertical lines. If it is true, lines are drawn as if their width is zero following L <sup>A</sup> T <sub>E</sub> X’s standard. Otherwise, their width <code>\arrayrulewidth</code> contribute to the width of columns as <code>array</code> does.

- `\ifadl@zwhrule` • `\ifadl@zwhrule` controls the *real* width of horizontal lines. If it is true, a line is drawn as if its width is zero and its bottom edge is adjusted to that of the row above by inserting `\vskip-\arrayulewidth` before the drawing. Thus a horizontal dash line is included in the row above and its gaps look colored properly if the row is painted. If it is false, the width `\arrayrulewidth` contribute to the height of `array/tabular` as usual.
- `\ifadl@usingarypkg` • `\ifadl@usingarypkg` is true iff `array` has been loaded prior to `arydshln`. This switch shows us which definitions, by `LATEX` or `array`, we have to modify. Its value is set by examining if `\extrarowheight`, which is introduced by `array`, is defined.

- `\ifadl@inactive` • `\ifadl@inactive` inactivates dash-line functions if it is true. Its default value is false.

We also use a working switch `\@tempwa`.

```

8 \newif\ifadl@leftrule
9 \newif\ifadl@connected
10 \newif\ifadl@doublerule
11 \newif\ifadl@zwvrule
12 \newif\ifadl@zwhrule
13 \newif\ifadl@usingarypkg
14 \ifx\extrarowheight\undefined \adl@usingarypkgfalse
15 \else \adl@usingarypkgtrue \fi
16 \newif\ifadl@inactive \adl@inactivefalse
17
```

- `\ADLnullwide` The switch `\ifadl@hwvrule` is turned on/off by user interface macros `\ADLnullwide` and
- `\ADLsomewide` `\ADLsomewide`. Its initial value is the complement of `\adl@usingarypkg`.
- `\ADLnullwidehline` The switch `\ifadl@zwvrule` is turned on/off by user interface macros `\ADLnullwidehline`
- `\ADLsomewidehline` and `\ADLsomewidehline`. Its initial value is false.
- `\ADLactivate` The switch `\ifadl@inactive` is also turned on/off by user interface macros `\ADL`
- `\ADLinactivate` `inactivate` and `\ADLactivate`.

```

18 \def\ADLnullwide{\adl@zwvruletrue}
19 \def\ADLsomewide{\adl@zwvrulefalse}
20 \ifadl@usingarypkg \ADLsomewide \else \ADLnullwide \fi
21 \def\ADLnullwidehline{\adl@zwhruletrue}
22 \def\ADLsomewidehline{\adl@zwhrulefalse}
23 \ADLsomewidehline
24
25 \def\ADLactivate{\adl@inactivefalse}
26 \def\ADLinactivate{\adl@inactivetrue}
27
```

The following `\box` register and three `\dimen` registers are used to measure the height and depth of a row.

- `\adl@box` • The contents of a column is packed into the `\box` register `\adl@box` to measure its height and depth.

<code>\adl@height</code> <code>\adl@depth</code>	<ul style="list-style-type: none"> <li>• The <code>\dimen</code> registers <code>\adl@height</code> and <code>\adl@depth</code> contain the height/depth of the tallest/deepest column in a row. When a column is processed, they are compared to the height and depth of <code>\adl@box</code> and are updated if they are less.</li> </ul>
<code>\adl@heightsave</code> <code>\adl@depthsave</code>	<p>Since we have to update these registers <code>\global</code>-ly to pass their values across <code>&amp;</code> and we may have a column containing <code>array/tabular</code>, they are saved into <code>\adl@heightsave/\adl@depthsave</code> at the beginning of the environment and are restored at its end.</p>
<code>\adl@finaldepth</code>	<ul style="list-style-type: none"> <li>• The other <code>\dimen</code> register <code>\adl@finaldepth</code> is set to the depth of the last row, or zero if the last vertical item is a horizontal line. This value is used to shift <code>array/tabular</code> down because we add extra two <code>\smash</code>-ed rows which make the depth of <code>array/tabular</code> zero.</li> </ul>

We also use working `\dimen` registers `\@tempdima` and `\@tempdimb`.

```

28 \newbox\adl@box
29 \newdimen\adl@height \newdimen\adl@heightsave
30 \newdimen\adl@depth \newdimen\adl@depthsave
31 \newdimen\adl@finaldepth

```

Then the following `\count` registers are declared. Note that some of them contain dimensions measured by the unit `sp`.

<code>\adl@columns</code> <code>\adl@ncol</code>	<ul style="list-style-type: none"> <li>• <code>\adl@columns</code> has the number of columns specified in the preamble of the environment. Because of a complicated reason related to the compatibility with <code>array</code>, we cannot count up <code>\adl@columns</code> directly but increment <code>\adl@ncol</code> when each column of preamble is built and move its value to <code>\adl@columns</code> after the preamble is constructed.</li> </ul>
<code>\adl@currentcolumn</code> <code>\adl@currentcolumnsave</code>	<ul style="list-style-type: none"> <li>• To process <code>\multicolumn</code>, we have to know the column number where it appears. Thus we have a column counter <code>\adl@currentcolumn</code> which is <code>\global</code>-ly incremented when each column is built. Because of the <code>\global</code> assignment, the counter has to be saved/restored into/from <code>\adl@currentcolumnsave</code>.</li> </ul>
<code>\adl@totalheight</code>	<ul style="list-style-type: none"> <li>• In the real implementation, <math>\tau_k</math> and <math>\beta_k</math> are calculated by the following equations rather than those shown in §4.1.</li> </ul>

$$H = \sum_{l=1}^N h_l, \quad \tau_k = H + \eta_{i-1} - \sum_{l=1}^{i-1} h_l, \quad \beta_k = \tau_k - \eta_{i-1} - \eta_{i'-1} - \sum_{l=i'}^i h_l.$$

`\adl@totalheight` contains  $\sum_{l=1}^i h_l$  when the  $i^{th}$  row is built and thus its final value is  $H$ . Since the data structure  $R$  are represented by a text, we have to pay attention to the precision of its dimensional elements, such as  $h_i$ . That is, if we append  $h_i$  to  $R$  by expanding `\the\dimenn` which has the height plus depth of  $i^{th}$  row,  $h_i$  will be an approximation of `\dimenn` represented by a decimal fraction with `pt`. Although the error of the approximation is quite small and may be negligible, the error must be avoided because it is avoidable by simply using `\number\dimenn`. Therefore,  $h_i$  is an integer and thus `\adl@totalheight` is too.



<code>\adl@totalheightsave</code>	Because of the <code>\global</code> assignment to <code>\adl@totalheight</code> to pass its value across rows, it has to be saved/restored into/from <code>\adl@totalheightsave</code> .
<code>\adl@dash</code> <code>\adl@gap</code>	• In order to check $e_j^i \sim e_{j'}^{i'}$ , the size attributes of $d_j^i$ and $g_j^i$ are kept in the registers <code>\adl@dash</code> and <code>\adl@gap</code> when we process $e_{j'}^{i'}$ . As explained above, $d_j^i$ and $g_j^i$ are integers and thus <code>\adl@dash</code> and <code>\adl@gap</code> are <code>\count</code> registers.
<code>\adl@cla</code> <code>\adl@clb</code>	• The coding of <code>\cdashline</code> is similar to that of <code>\cline</code> in L <sup>A</sup> T <sub>E</sub> X-2.09 which uses two global <code>\count</code> registers <code>\@cla</code> and <code>\@clb</code> . These registers are omitted from L <sup>A</sup> T <sub>E</sub> X 2 <sub>ε</sub> because its <code>\cline</code> is completely recoded. We could adopt new coding but it requires some other macro definitions that L <sup>A</sup> T <sub>E</sub> X-2.09 does not have. Thus we simply introduce new global counters <code>\adl@cla</code> and <code>\adl@clb</code> for <code>\cdashline</code> in order to make <code>\cdashline</code> work in both L <sup>A</sup> T <sub>E</sub> X-2.09 and L <sup>A</sup> T <sub>E</sub> X 2 <sub>ε</sub> .

We also use working `\count` registers `\@tempcnta` and `\@tempcntb`.

```

32 \newcount\adl@columns \newcount\adl@ncol
33 \newcount\adl@currentcolumn \newcount\adl@currentcolumnsave
34 \newcount\adl@totalheight \newcount\adl@totalheightsave
35 \newcount\adl@dash \newcount\adl@gap
36 \newcount\adl@cla \newcount\adl@clb

```

`\adl@everyvbox` The last register declaration is for a `\toks` register named `\adl@everyvbox`. In order to minimize the copy-and-modify of the codes in L<sup>A</sup>T<sub>E</sub>X and `array`, we need to use `\everyvbox` in our own definition of `\@array`. The register is used to save the contents of `\everyvbox`.

```

37 \newtoks\adl@everyvbox
38

```

<code>\adl@org@arrayclassz</code>	The other declarative stuff consists of the sequence of <code>\let</code> to capture the original definitions of macros that we will modify afterword. The main purpose of them is to nullify the modification when dash-line functions are inactive, while <code>\adl@org@cline</code> is also referred to in its modified version.
<code>\adl@org@tabclassz</code>	
<code>\adl@org@classz</code>	
<code>\adl@org@@startpbox</code>	
<code>\adl@org@@endpbox</code>	
<code>\adl@org@cline</code>	

```

39 \let\adl@org@arrayclassz\@arrayclassz
40 \let\adl@org@tabclassz\@tabclassz
41 \let\adl@org@classz\@classz
42 \let\adl@org@@startpbox\@startpbox
43 \let\adl@org@@endpbox\@endpbox
44 \let\adl@org@endpbox\@endpbox
45 \let\adl@org@cline\cline
46
47 %^\L

```

## 4.4 Initialization

<code>\adl@array</code>	L <sup>A</sup> T <sub>E</sub> X's macro <code>\@array</code> is modified to save and initialize registers and data structures which are <code>\global</code> -ly updated in order to allow nested <code>array/tabular</code> . This saving and initializing are performed by <code>\adl@arrayinit</code> as explained below. The problem in the
<code>\@array</code>	
<code>\adl@noalign</code>	

modification is that the code of `\@array` in `array` is completely different from that of L<sup>A</sup>T<sub>E</sub>X original.

The main difference is that L<sup>A</sup>T<sub>E</sub>X builds `\@preamble` locally, while `array` does globally exploiting the fact that the lifetime of `\@preamble` ends before another `array/tabular` appears in a column. The latter implementation will work well unless the building process in `\@mkpream` produces something referred to after `\@preamble` is thrown into T<sub>E</sub>X's *stomach*. In our implementation, unfortunately, the number of columns has to be counted in `\@mkpream` and will be referred to by `\hdashline` and the vertical line drawing procedure.

Thus we have to change the column counting mechanism depending on whether or not `array` is in use. The simplest way could be to copy the codes of L<sup>A</sup>T<sub>E</sub>X and `array` and modify them appropriately examining the value of `\ifadl@usingarypkg`. However this solution is vulnerable to the modification of the original version and thus we wish to refuse it as far as possible.

Therefore, we use a trick with `\everyvbox` in which `\adl@arrayinit` is temporarily included to initialize registers and locally set `\adl@columns` to the number of columns `\global`-ly counted by `\adl@ncol`. This trick works well so far because;

- the first `\vbox`, `\vtop` or `\vcenter` made by `\@array` is the vertical box surrounding `\halign`, and;
- in `\@array` of `array` the box is opened *after* the preamble is constructed;

and will hopefully work in future.

Next, if `\ifadl@inactive` is true, `\adl@inactivate` is invoked to inactivate dash-line functions. Otherwise, `\adl@activate` is invoked to activate them because an inactivated `array/tabular` may have active children in it. Finally, `\adl@noalign` is made `\let`-equal to `\noalign` so that `\arrayrulecolor`, `\doublerulesepcolor` and `\dashgapcolor` are expanded with `\noalign` in the environment.

`\@@array` Another stuff for the compatibility with `array` is to `\let` a control sequence `\@@array` be equal to `\@array` if it is made so by `array` and the equality is kept. That is, with `array` `\@@array` is invoked by `\@tabarray` and it is `\let`-equal to `\@array` by default, while `\@@array` can be made different from `\@array` by some other package, e.g., `delarray`, to do some special operations defined in the package. Therefore by the conditional equalization with `\ifx`, our own `\@array` is directly invoked through `\@@array` if the default equality is kept, while otherwise the package-dependent definition of `\@@array` is respected.

```

48
49 %% Initialization
50
51 \let\adl@array\@array
52 \def\@array{\adl@everyvbox\everyvbox
53     \everyvbox{\adl@arrayinit \the\adl@everyvbox \everyvbox\adl@everyvbox}%
54     \ifadl@inactive \adl@inactivate \else \adl@activate \fi
55     \let\adl@noalign\noalign
56     \adl@array}
57 \ifx\@@array\adl@array \let\@@array\@array \fi
58

```

`\adl@arrayinit` As described in §4.3, registers updated `\global`-ly, which are `\adl@height`, `\adl@depth`,  
`\adl@arraysave` `\adl@currentcolumn` and `\adl@totalheight`, are saved in `\adl@arrayinit` by calling  
`\adl@arraysave`, and also given initial values. The macro also saves the following data  
structures and initializes them to empty lists.

`\adl@rowsL` • In the real implementation, the data structure  $R$  is split into two lists;  
`\adl@rowsR`  
`\adl@rowsLsave`  $\text{\adl@rowsL} = R^L = \langle \langle C_1^L, h_1 \rangle, \dots \rangle$   
`\adl@rowsRsave`  $\text{\adl@rowsR} = R^R = \langle \langle C_1^R, h_1 \rangle, \dots \rangle$   
`\adl@colsL`  
`\adl@colsR` and they are saved into `\adl@rowsLsave` and `\adl@rowsRsave`.  
`\adl@colsLsave` • When the  $i^{\text{th}}$  row is building,  $C_i^L$  and  $C_i^R$  are constructed in the macros `\adl@colsL`  
`\adl@colsRsave` and `\adl@colsR`. They are saved into `\adl@colsLsave` and `\adl@colsRsave`.

`\adl@connect` In the real implementation,  $e_j^i$  is represented by a control sequence `\@elt`, and `connect(i)` by  
`\adl@discard` `\adl@connect`. They are made `\let`-equal to `\relax` to keep them from expansion during  $R$   
is constructed. In `longtable` environment, `connect(i)` for negative vertical space inserted  
by `\[\langle h \rangle]` or a horizontal line has another representation `\adl@discard` to indicate it  
corresponds to a discardable item of page breaking. Since this representation, however,  
is nonsense in usual `array`/`tabular` even if they are included in `\longtable`, we define  
`\adl@discard` as `\adl@connect` so that it transforms itself into `\adl@connect` when it is  
added to `\adl@rowsL/R` by `\xdef`. Note that `\adl@discard` is made `\let`-equal to `\relax`  
to inhibit the transformation at the beginning of `longtable` environment.

Then, we set to `\adl@columns` to the value of `\adl@ncol` *locally*. As explained above,  
this has an effect with `array` because `\adl@arrayinit` is called *after* the preamble is gener-  
ated. Without `array`, on the other hand, this assignment has no effect but safe because it is  
included in a group of `\vbox` etc.

```

59 \def\adl@arrayinit{%
60     \adl@arraysave
61     \global\adl@height\z@ \global\adl@depth\z@
62     \global\adl@currentcolumn\@ne \global\adl@totalheight\z@
63     \gdef\adl@rowsL{}\gdef\adl@rowsR{}\gdef\adl@colsL{}\gdef\adl@colsR{}%
64     \let\@elt\relax \let\adl@connect\relax \def\adl@discard{\adl@connect}%
65     \adl@columns\adl@ncol}
66 \def\adl@arraysave{%
67     \adl@heightsave\adl@height
68     \adl@depthsave\adl@depth
69     \adl@currentcolumnsave\adl@currentcolumn
70     \adl@totalheightsave\adl@totalheight
71     \let\adl@rowsLsave\adl@rowsL
72     \let\adl@rowsRsave\adl@rowsR
73     \let\adl@colsLsave\adl@colsL
74     \let\adl@colsRsave\adl@colsR}
75

```

`\adl@inactivate` If `\ADLinactivate` has effect and thus `\ifadl@inactive` is true, the macro `\adl@inactivate` is called from `\@array`<sup>13</sup>. This `\let`-s the following control sequences be equal to their counterparts in L<sup>A</sup>T<sub>E</sub>X and/or array package.

```
\@arrayclassz \@tabclassz \@classz @@startpbox @@endpbox
\endpbox \adl@cr \adl@argcr \adl@endarray
```

Note that we have to inactivate both `@@endpbox` for L<sup>A</sup>T<sub>E</sub>X and `\endpbox` for array, while `\startpbox` for array is not necessary because it is unmodified. Also note that `\@classz` has to be `\let`-equal to `\adl@org@classz` only if array is in use, because L<sup>A</sup>T<sub>E</sub>X does not define `\@classz` but refers to it which is either `\@arrayclassz` or `\@tabclassz`. Yet another remark is that we have to conceal `\cr` for `\adl@cr/\adl@argcr` and `\crrcr` for `\adl@endarray` by bracing them from T<sub>E</sub>X's `\halign` mechanism that searches them when an `array/tabular` has an nested `array/tabular`. This could be done by a tricky `\let`-assignment such as;

```
\iffalse{\let\adl@cr\cr \iffalse}\fi
```

but we simply use `\def` instead of `\let` because of clarity.

We also `\let` the following be *no-operation* or their inactive versions.

```
\adl@hline \adl@ihdashline \adl@cdline \adl@@vlineL \adl@@vlineR
\adl@vlineL \adl@vlineR
```

Note that we have to inactivate both `\adl@@vlineL` and `\adl@vlineL`, because the latter is referred to when array is in use while the former is done otherwise. Their R relatives are also inactivated by the same reason.

```
76 \def\adl@inactivate{%
77     \let\@arrayclassz\adl@org@arrayclassz
78     \let\@tabclassz\adl@org@tabclassz
79     \ifadl@usingarypkg \let\@classz\adl@org@classz \fi
80     \let\@@startpbox\adl@org@@startpbox
81     \let\@@endpbox\adl@org@@endpbox
82     \let\@endpbox\adl@org@endpbox
83     \def\adl@cr{\cr}%
84     \def\adl@argcr##1{\cr}%
85     \def\adl@endarray{\crrcr}%
86     \let\adl@hline\@gobbletwo
87     \let\adl@ihdashline\adl@inactivehdl
88     \let\adl@cdline\adl@inactivecdl
89     \let\adl@@vlineL\adl@inactivevl
90     \let\adl@@vlineR\adl@inactivevl
91     \let\adl@vlineL\adl@inactivevl
92     \let\adl@vlineR\adl@inactivevl}
```

`\adl@activate` On the other hand, if `\ifadl@inactive` is false, the macro `\adl@activate` is called from `\@array` to make inactivated macros active again in order to cope with the case in which an inactive `array/tabular` has active children in it<sup>14</sup>. To do that, `\adl@activate` makes

<sup>13</sup>Before v1.53, `\adl@inactivate` was called from `\adl@arrayinit` and thus invoked *after* the preamble of `array` is built. This was incorrect of course and made inactive version of `p`, `m` and `b` produce nothing.

<sup>14</sup>Before v1.54, an active `array/tabular` in an inactive parent was not activated.

Table 1: Active and Inactive Operations

command	active	inactive
<code>l c r</code>		
with array	<code>\adl@act@classz</code>	<code>\adl@org@classz</code>
without array	<code>\adl@act@tabclassz</code>	<code>\adl@org@tabclassz</code>
	<code>\adl@act@arrayclassz</code>	<code>\adl@org@arrayclassz</code>
<code>p m b</code> (open)		
with array	<code>\adl@act@classz</code>	<code>\adl@org@classz</code>
without array	<code>\adl@act@@startpbox</code>	<code>\adl@org@@startpbox</code>
<code>p m b</code> (close)	<code>\adl@act@@endpbox</code>	<code>\adl@org@@endpbox</code>
<code> /:/;</code>	<code>\adl@act@@vlineL/R</code>	<code>\adl@inactivevl</code>
<code>\</code>	<code>→\adl@act@(arg)cr</code>	<code>→\cr</code>
<code>\hline</code>	<code>→\adl@act@hline</code>	<code>→\gobbletwo</code>
<code>\hdashline</code>	<code>→\adl@act@ihdashline</code>	<code>→\adl@inactivehdl</code>
<code>\cdashline</code>	<code>→\adl@act@cdline</code>	<code>→\adl@inactivecdl</code>

`\@arrayclassz` etc. `\let`-equal to their active version `\adl@act@arrayclassz` etc. which will be defined (`\let`-equal to) as our own `\@arrayclassz` etc. in §4.13.

```

93 \def\adl@activate{%
94     \let\@arrayclassz\adl@act@arrayclassz
95     \let\@tabclassz\adl@act@tabclassz
96     \ifadl@usingarypkg \let\@classz\adl@act@classz \fi
97     \let\@@startpbox\adl@act@@startpbox
98     \let\@@endpbox\adl@act@@endpbox
99     \let\@endpbox\adl@act@endpbox
100    \let\adl@cr\adl@act@cr
101    \let\adl@argcr\adl@act@argcr
102    \let\adl@endarray\adl@act@endarray
103    \let\adl@hline\adl@act@hline
104    \let\adl@ihdashline\adl@act@ihdashline
105    \let\adl@cdline\adl@act@cdline
106    \let\adl@@vlineL\adl@act@@vlineL
107    \let\adl@@vlineR\adl@act@@vlineR
108    \let\adl@vlineL\adl@act@@vlineL
109    \let\adl@vlineR\adl@act@@vlineR}
110
111 %%^L

```

The summary of the activation and inactivation is shown in Table 1.

## 4.5 Making Preamble

Each preamble character is converted to a part of `\halign`'s preamble as follows.

`\adl@colhtdp` • ‘l’, ‘r’ and ‘c’ are converted to the following  $\langle lrc \rangle$ .

$$\begin{aligned}\langle lrc \rangle &::= [\hfil]\langle put-lrc \rangle[\hfil] \\ \langle put-lrc \rangle &::= \setbox\adl@box\hbox{\langle lrc-contents \rangle} \\ &\quad \adl@colhtdp \unhbox\adl@box \\ \langle lrc-contents \rangle &::= \$\relax\#\$ | \\ &\quad \#\unskip\end{aligned}$$

That is, the content of a column is at first packed into the `\box` register `\adl@box`, then its height and depth are compared to `\adl@height` and `\adl@depth` by the macro `\adl@colhtdp`, and finally the box is put with leading and/or trailing `\hfil`.

`\adl@vlineL` • ‘|’, ‘:’ and  $\langle dash \rangle / \langle gap \rangle$  are converted to the following  $\langle vline \rangle$ .  
`\adl@vlineR`

$$\begin{aligned}\langle vline \rangle &::= [\hskip\doublerulesep]\langle vline-LR \rangle \\ \langle vline-LR \rangle &::= \adl@vlineL\{\langle \Gamma_d \rangle\}\{\langle \Gamma_g \rangle\}\{\langle c \rangle\}\{\langle d \rangle / \langle g \rangle\} | \\ &\quad \adl@vlineR\{\langle \Gamma_d \rangle\}\{\langle \Gamma_g \rangle\}\{\langle c \rangle\}\{\langle d \rangle / \langle g \rangle\} \\ \langle d \rangle &::= 0 | \dots \text{for ‘|’} \\ &\quad \dashlinedash | \dots \text{for ‘:’} \\ &\quad \langle dash \rangle | \dots \text{for ‘;’} \\ \langle g \rangle &::= 0 | \dots \text{for ‘|’} \\ &\quad \dashlinegap | \dots \text{for ‘:’} \\ &\quad \langle gap \rangle | \dots \text{for ‘;’}\end{aligned}$$

Note that  $\langle c \rangle$  is the column number (leftmost is 1) where the character appears, and  $\langle \Gamma_d \rangle$  and  $\langle \Gamma_g \rangle$  is the color of dashes and gaps specified in `\CT@arc@` and `\adl@dashgapcolor`.

Additionally, each column except for the last one has;

$$\global\advance\adl@currentcolumn\@ne$$

just before `&` to increment `\adl@currentcolumn`. Other features, such as inserting spaces of `\arraycolsep`/`\tabcolsep`, are as same as original scheme. This means that  $\mathcal{O}\{\langle text \rangle\}$  and  $!\{\langle text \rangle\}$  of `array` are *not* handled specially although it could interfere with drawing vertical lines. Therefore, we have the problem 1 shown in §3, which is very hard to solve. Note that the measurement of the column of ‘p’ of L<sup>A</sup>T<sub>E</sub>X original is done by (modified) `\@@startpbox` and `\@@endpbox` and thus the preamble for ‘p’ is not modified. In the case with `array`, however, the preambles for ‘p’ and its relatives ‘m’ and ‘b’ are modified to set `\adl@box` to the box for them.

`\adl@mkpream` To make the preamble shown above, `\@mkpream` is modified to `\let` control sequences  
`\@mkpream` `\adl@colhtdp`, `\adl@vlineL` and `\adl@vlineR` be `\relax` in order to keep them from being expanded by `\edef`/`\xdef` for the preamble construction. The control sequences

`\adl@startmbox` and `\adl@endmbox` for `m`-columns of `array` are also made `\let`-equal to `\relax`.

Giving them their own definition is done by `\adl@preamininit` that is called using `\afterassignment` after `\@preamble` is made by `\adl@mkpream`, the original version of `\@mkpream`. If `array` is not in use, `\@mkpream` is followed by an `\edef` of `\@preamble` to add `\ialign` etc. and thus `\adl@preamininit` is properly called *after* this final *assignment* to make `\@preamble`.

With `array`, on the other hand, calling `\adl@preamininit` is safe because `\@mkpream` is followed by `\xdef` for `\@preamble` too, but has no effect because it is in the group for `\@mkpream`. This grouping, however, gives us an easier way to give those control sequences their own definition. That is, we simply initiate them with the definitions that will be regained when the group is closed.

The modified `\@mkpream` also initializes `\adl@ncol` and `\ifadl@leftrule`, and set `\adl@columns` to the value of `\adl@ncol` locally after the preamble is made. This has an effect in the case without `array` because the body of `array/tabular` is in the same grouping context of `\@mkpream`. With `array`, on the other hand, this assignment has no effect but safe because it is included in a group of `\@mkpream`'s own.

```

112
113 %% Making Preamble
114
115 \let\adl@mkpream\@mkpream
116 \def\@mkpream#1{\let\adl@colhtdp\relax
117     \let\adl@vlineL\relax \let\adl@vlineR\relax
118     \let\adl@startmbox\relax \let\adl@endmbox\relax
119     \global\adl@ncol\@ne \adl@leftruletrue
120     \adl@mkpream{#1}\adl@columns\adl@ncol \afterassignment\adl@preamininit}
121

```

`\@addamp` The macro `\@addamp` is also modified to add the code for incrementing the counter `\adl@currentcolumn` to `\@preamble` with `&`. The counter `\adl@ncol` is also incremented by `\@addamp` so that we can refer to its value as `\c` of `\adl@vlineL/R`. This increment is done `\global`-ly in order that we locally set `\adl@columns` to the counting result outside of the group for `\@mkpream` of `array`. Therefore, whether or not `array` is in use, `\adl@columns` will have a correct value and will be correctly referred to by `\hdashline` to know how many columns are specified in the preamble. Note that this `\global` assignment is safe because the life time of `\adl@ncol` is same as that of `\@preamble`.

```

122 \def\@addamp{\if@firstamp\@firstampfalse \else
123     \@addtopreamble{\global\advance\adl@currentcolumn\@ne &}}%
124     \global\advance\adl@ncol\@ne \fi}
125

```

Since the implementation of `\@testpach` and macros for class-0 characters (i.e. `l`, `r` and `c`) is completely different between  $\text{\LaTeX}$  and `array`, we have to have two versions switched by `\adl@usingarypkg`.

## With array

`\@testpach` Although we introduced two preamble characters ‘:’ and ‘;’, we did not introduce new character *class* because we want to minimize the modification of original codes. Therefore, ‘:’ and ‘;’ is classified into class-1 together with ‘|’. Since these characters obviously have their own appropriate operations, `\@testpach` is modified so that `\@arrayrule`, which is invoked from `\@mkpream` in the case of class-1 character, is `\let`-equal to the macro corresponding to each character.

```

126 \ifadl@usingarypkg
127 \def\@testpach{\@chclass
128   \ifnum \@lastchclass=6 \@ne \@chnum \@ne \else
129     \ifnum \@lastchclass=7 5 \else
130       \ifnum \@lastchclass=8 \tw@ \else
131         \ifnum \@lastchclass=9 \thr@@
132         \else \z@
133         \ifnum \@lastchclass = 10 \else
134           \edef\@nextchar{\expandafter\string\@nextchar}%
135           \@chnum
136           \if \@nextchar c\z@ \else
137             \if \@nextchar l\@ne \else
138               \if \@nextchar r\tw@ \else
139               \z@ \@chclass
140               \if \@nextchar |\@ne \let\@arrayrule\adl@arrayrule \else
141               \if \@nextchar : \@ne \let\@arrayrule\adl@arraydashrule \else
142               \if \@nextchar ; \@ne \let\@arrayrule\adl@argarraydashrule \else
143               \if \@nextchar !6 \else
144               \if \@nextchar @7 \else
145               \if \@nextchar <8 \else
146               \if \@nextchar >9 \else
147               10
148               \@chnum
149               \if \@nextchar m\thr@@\else
150               \if \@nextchar p4 \else
151               \if \@nextchar b5 \else
152               \z@ \@chclass \z@ \@preamerr \z@ \fi \fi \fi \fi \fi \fi
153               \fi \fi \fi \fi \fi \fi \fi \fi \fi \fi \fi \fi
154
```

`\@classz` In array, array and tabular share common macro for class-0 named `\@classz`, which also generates the preamble for ‘p’, ‘m’ and ‘b’. Thus we modify it to measure the height and depth of the class-0 column by the macro `\adl@putlrc`, and to set `\adl@box` to the box for ‘p’ and its relatives. Note that a m-type preamble (`@chnum = 3`) has to be generated to have `\adl@startmbox` and `\adl@endmbox` in it because a `\vcenter` construct cannot be assigned to `\adl@box` by `\setbox` directly.

```

155 \def\@classz{\@classx
156   \@tempcnta \count@
157   \prepnext@tok
158   \@addtopreamble{\ifcase \@chnum
```



```

159      \hfil
160      \adl@putlrc{\d@llarbegin \insert@column \d@llarend}\hfil \or
161      \hskip1sp\adl@putlrc{\d@llarbegin \insert@column \d@llarend}\hfil \or
162      \hfil\hskip1sp\adl@putlrc{\d@llarbegin \insert@column \d@llarend}\or
163      \setbox\adl@box\hbox \adl@startmbox{\@nextchar}\insert@column
164      \adl@endmbox\or
165      \setbox\adl@box\vtop \@startpbox{\@nextchar}\insert@column \@endpbox \or
166      \setbox\adl@box\vbox \@startpbox{\@nextchar}\insert@column \@endpbox
167      \fi}\prepnext@tok}

```

`\adl@class@start` Another stuff for compatibility is to refer to the class number for the beginning of preamble which is different between L<sup>A</sup>T<sub>E</sub>X and `array`, and that for ‘p’ or ‘@’ to get the argument of ‘;’ as explained later. In the case with `array`, the former is class-4 and we use ‘@’ (class-7) for the latter.

```

168 \def\adl@class@start{4}
169 \def\adl@class@iiiorvii{7}
170

```

### Without `array`

`\@testpach` The reason why and how we modify `\@testpach` of L<sup>A</sup>T<sub>E</sub>X is same as those of `array`.

```

171 \else
172 \def\@testpach#1{\@chclass \ifnum \@lastchclass=\tw@ 4\relax \else
173     \ifnum \@lastchclass=\thr@@ 5\relax \else
174         \z@ \if #1c\@chnum \z@ \else
175             \if #1l\@chnum \@ne \else
176                 \if #1r\@chnum \tw@ \else
177                     \@chclass
178                     \if #1|\@ne \let\@arrayrule\adl@arrayrule \else
179                         \if #1:\@ne \let\@arrayrule\adl@arraydashrule \else
180                             \if #1;\@ne \let\@arrayrule\adl@argarraydashrule \else
181                                 \if #1@\tw@ \else
182                                     \if #1p\thr@@ \else \z@ \@preamerr 0\fi
183                             \fi \fi \fi \fi \fi \fi \fi \fi \fi \fi
184

```

`\@arrayclassz` Since L<sup>A</sup>T<sub>E</sub>X has two macros for class-0, one for `array` and the other for `tabular`, we have to modify both. Since the box for ‘p’ is opened by `\@startpbox`, however, we may not worry about it.

```

185 \def\@arrayclassz{\ifcase \@lastchclass \@acolampacol \or \@ampacol \or
186     \or \or \@addamp \or
187     \@acolampacol \or \@firststampfalse \@acol \fi
188 \edef\@preamble{\@preamble
189     \ifcase \@chnum
190         \hfil\adl@putlrc{\$relax\@sharp$}\hfil
191         \or \adl@putlrc{\$relax\@sharp$}\hfil
192         \or \hfil\adl@putlrc{\$relax\@sharp$}\fi}}

```

```

193 \def\@tabclassz{\ifcase \@lastchclass \@acolampacol \or \@ampacol \or
194         \or \or \@addamp \or
195         \@acolampacol \or \@firstampfalse \@acol \fi
196 \edef\@preamble{\@preamble
197 \ifcase \@chnum
198         \hfil\adl@putlrc{\@sharp\unskip}\hfil
199         \or \adl@putlrc{\@sharp\unskip}\hfil
200         \or \hfil\hskip\z@ \adl@putlrc{\@sharp\unskip}\fi}}

```

`\adl@class@start` In L<sup>A</sup>T<sub>E</sub>X, the beginning of preamble is class-6 and we use ‘p’ (class-3) to get the argument of ‘;’.

```

201 \def\adl@class@start{6}
202 \def\adl@class@iiiorvii{3}
203 \fi
204

```

Hereafter, codes for L<sup>A</sup>T<sub>E</sub>X and array are common again.

`\adl@putlrc` The macro `\adl@putlrc` is for class-0 preamble characters to set `\adl@box` to the contents of a column, measure its height/depth by `\adl@colhtdp` and put the box by `\unhbox` (not by `\box`) in order to make the glues in the contents effective.

```

205 \def\adl@putlrc#1{\setbox\adl@box\hbox{#1}\adl@colhtdp \unhbox\adl@box}
206

```

`\adl@arrayrule` The preamble parts for vertical solid- and dash-lines are constructed by the macros `\adl@arrayrule` for ‘|’, `\adl@arraydashrule` for ‘:’, and `\adl@argarraydashrule` for ‘;’. The macro;

```

\adl@xarraydashrule
\adl@xarraydashrule{\langle c^L \rangle}{\langle c^R \rangle}{\langle d \rangle / \langle g \rangle}

```

is invoked by them to perform common operations. It at first checks the preamble character is the first element of the preamble (`\@lastchclass = \adl@class@start`) or it follows another character for vertical line (`\@lastchclass = 1`). If this is not satisfied, the vertical line is put at the right edge of a column and thus `\ifadl@leftrule` is set to false. Then it adds `\adl@vlineL{\langle \Gamma_d \rangle}{\langle \Gamma_g \rangle}{\langle c^L \rangle}{\langle d \rangle / \langle g \rangle}` if `\ifadl@leftrule` is true indicating the vertical line will appear at the left edge of the column  $\langle c^L \rangle$ , or `\adl@vlineR{\langle \Gamma_d \rangle}{\langle \Gamma_g \rangle}{\langle c^R \rangle}{\langle d \rangle / \langle g \rangle}` otherwise. Note that  $\langle c^L \rangle$  is always 1 for *main* preamble while  $\langle c^R \rangle$  is the column number given by `\adl@ncol`, but  $\langle c^L \rangle$  may not be 1 for the preamble of `\multicolumn` as described in §4.7. Also note that  $\Gamma_d$  and  $\Gamma_g$  are `\CT@arc@` and `\adl@dashgapcolor` respectively whose bodies are `\color` for dashes and gaps specified by `\arrayrulecolor` and `\dashgapcolor`, or `\relax` if they are not colored.

In addition, an invisible `\vrule` of `\arrayrulewidth` wide is added if both `\ADLsome wide` and `\ADLactivate` are in effect, i.e. both `\ifadl@zwrule` and `\ifadl@inactive` are false, to keep a space for the vertical line having *real* width.

`\adl@classv` The argument of ‘;’ is not provided by `\adl@argarraydashrule` but is directly passed from the preamble text through `\@nextchar`. This direct passing is implemented by the following trick. The macro `\adl@argarraydashrule` set `\@chclass` to `\adl@class@iiiorvii` to

pretend it is for ‘p’ if `array` is not in use, or ‘@’ otherwise. Then it temporally changes the definition of `\@classv`, which is incidentally for the argument of ‘p’ and ‘@’ in the case without/with `array` respectively, to `\adl@classvfordash` to process the argument of ‘;’ rather than that of ‘p’ or ‘@’. Then `\adl@classvfordash` is invoked by `\@mkpream` and it adds the argument to `\@preamble`. Finally, it restores the definition of `\@classv` and sets `\@chclass` to 1 to indicate that the last item is a vertical line specification.

```

207 \def\adl@arrayrule{%
208     \adl@xarraydashrule
209     {\@ne}{\adl@ncol}{\z@/\z@}}
210 \def\adl@arraydashrule{%
211     \adl@xarraydashrule
212     {\@ne}{\adl@ncol}%
213     {\dashlinedash/\dashlinegap}}
214 \def\adl@argarraydashrule{%
215     \adl@xarraydashrule
216     {\@ne}{\adl@ncol}{}%
217     \@chclass\adl@class@iiiorvii \let\@classv\adl@classvfordash}
218 \def\adl@xarraydashrule#1#2#3{%
219     \ifnum\@lastchclass=\adl@class@start\else
220     \ifnum\@lastchclass=\@ne\else
221         \adl@leftrulefalse \fi\fi
222     \ifadl@zwvrule\else \ifadl@inactive\else
223         \@addtopreamble{\vrule\@width\arrayrulewidth
224             \@height\z@ \@depth\z@}\fi \fi
225     \ifadl@leftrule
226         \@addtopreamble{\adl@vlineL{\CT@arc@}{\adl@dashgapcolor}%
227             {\number#1}\#3}%
228     \else \@addtopreamble{\adl@vlineR{\CT@arc@}{\adl@dashgapcolor}%
229         {\number#2}\#3}\fi}
230 \let\adl@classv\@classv
231 \def\adl@classvfordash{\@addtopreamble{\@nextchar}}\let\@classv\adl@classv
232     \@chclass\@ne}
233
234 %%^L

```

## 4.6 Building Columns

`\adl@preamininit` If `array` is not in use, after the `\@preamble` is completed, the control sequences for macros in `\adl@colhtdp` it should regain their own definition. The macro `\adl@preamininit` performs this operation for macros we introduced, `\adl@colhtdp`, `\adl@vlineL` and `\adl@vlineR`. For the case with `array`, we will call `\adl@preamininit` in `arydshln` to initiate them with the definitions as described later.

```

235
236 %% Building Columns
237
238 \def\adl@preamininit{\let\adl@colhtdp\adl@@colhtdp
239     \let\adl@vlineL\adl@@vlineL \let\adl@vlineR\adl@@vlineR}

```

`\adl@colhtdp` For the measurement of the height and depth of a row, `\adl@colhtdp` compares `\adl@height` and `\adl@depth` to the height and depth of `\adl@box` which contains the main part of the column to be built, and `\global`-ly updates the registers if they are less.

```

241 \def\adl@colhtdp{%
242     \ifdim\adl@height<\ht\adl@box \global\adl@height\ht\adl@box \fi
243     \ifdim\adl@depth<\dp\adl@box \global\adl@depth\dp\adl@box\fi}
244

```

`\adl@vlineL` The macro `\adl@vlineL` $\langle\Gamma_d\rangle\langle\Gamma_g\rangle\langle c\rangle\langle d\rangle\langle g\rangle$  adds the element  $e = \langle c, d, g \rangle = \text{\@elt}\{\langle c \rangle\}\{\langle d \rangle\}\{\langle g \rangle\}\{\langle \gamma_d \rangle\}\{\langle \gamma_g \rangle\}$  to the tail of the list `\adl@colsL` to construct  $C_i^L$ , where  $\gamma_d$  and  $\gamma_g$  are the color specifications given by `\color` macros in  $\Gamma_d$  and  $\Gamma_g$ . The macro `\adl@vlineR` performs similar operation but the element is added to the head of `\adl@colsR` for  $C_i^R$  because it is processed right-to-left manner. The argument  $\langle d \rangle$  and  $\langle g \rangle$  are extracted by the macro `\adl@ivline` which converts given dimensional values of them to integers. It also sets  $\langle d \rangle$  and  $\langle g \rangle$  to 0 (i.e. solid-line) if one of given values are not positive, in order to make it sure that one dash segment has positive length. Then it invokes `\adl@setcolor` to define `\adl@dashcolor` and `\adl@gapcolor` with the color specification of  $\Gamma_d$  and  $\Gamma_g$ . Since `\adl@setcolor` locally expands `\color` macro in  $\Gamma_d$  and  $\Gamma_g$  to define `\currentcolor` that becomes the body of `\adl@dashcolor` ( $\gamma_d$ ) and `\adl@gapcolor` ( $\gamma_g$ ) with expansion, different `\color` specifications of a color, such as `\color{red}` and `\color[rgb]{1,0,0}`, will produce a unified result such as `{rgb 1 0 0}`. If  $\Gamma_d$  or  $\Gamma_g$  is `\relax` which is the body of `\adl@nocolor`,  $\gamma_d$  or  $\gamma_g$  is also `\relax` to indicate dashes are colored (or not colored) as done in outer world and gaps are transparent.

```

245 \def\adl@vlineL#1#2#3#4{\adl@ivline#4\@nil{#1}{#2}}%
246     \xdef\adl@colsL{\adl@colsL
247         \@elt{#3}{\number\@tempcnta}{\number\@tempcntb}%
248         {\adl@dashcolor}{\adl@gapcolor}}%
249 \def\adl@vlineR#1#2#3#4{\adl@ivline#4\@nil{#1}{#2}}%
250     \xdef\adl@colsR{%
251         \@elt{#3}{\number\@tempcnta}{\number\@tempcntb}%
252         {\adl@dashcolor}{\adl@gapcolor}%
253         \adl@colsR}%
254 \def\adl@ivline#1/#2\@nil#3#4{%
255     \@tempdima#1\relax \@tempcnta\@tempdima
256     \@tempdima#2\relax \@tempcntb\@tempdima
257     \ifnum\@tempcnta>\z@ \else \@tempcnta\z@ \@tempcntb\z@ \fi
258     \ifnum\@tempcntb>\z@ \else \@tempcnta\z@ \@tempcntb\z@ \fi
259     \adl@setcolor\adl@dashcolor{#3}\adl@setcolor\adl@gapcolor{#4}}
260 \def\adl@setcolor#1#2{\def\@tempa{#2}\ifx\@tempa\adl@nocolor \def#1{\relax}%
261     \else{#2\xdef#1{\currentcolor}}\fi}
262 \def\adl@nocolor{\relax}

```

`\adl@colhtdp` After `\adl@colhtdp`, `\adl@vlineL` and `\adl@vlineR` are defined, we call `\adl@preaminit` to `\let` their single `@` counterparts be equal to them. Therefore, in the case `\adl@vlineL` with array, `\adl@colhtdp` etc. are temporarily `\relax` when `\@preamble` is being generated

in the group of `\@mkpream`, and regain their own definitions outside the group where the completed `\@preamble` is referred to.

```

263 \adl@preamininit
264
\adl@inactivevl If \ADLinactivate is in effect, \adl@vlineL/R and \adl@@vlineL/R are \let-equal to
\adl@inactivevl. This macro simply puts a \vrule by \vline with \color (or \relax)
in its first argument and with/without negative \hskip of a half of \arrayrulewidth wide
depending on \ifadl@zwvrule, discarding other arguments.

265 \def\adl@inactivevl#1#2#3#4{\ifadl@zwvrule \hskip-.5\arrayrulewidth \fi
266      \{#1\vline}\ifadl@zwvrule \hskip-.5\arrayrulewidth \fi}
267

\@@startpbox The macros to make \parbox for ‘p’ (and ‘b’ of array), \@@startpbox and \@@endpbox,
\@@endpbox are modified for height/depth measurement. The code for \@@endpbox is based on that
\@endpbox of LATEX 2ε to fix the bug of \strut-ing in LATEX-2.09, but \@finalstrut is manually
\adl@startmbox expanded because it is not available in LATEX-2.09.
\adl@endmbox In array, \@@endpbox is not used but \@endpbox is. Therefore, we \let them be
equal. As for \@startpbox, however, we may not worry about it because we have modified
\@classz in §4.5 for the measurement. However, we have to take care of m-type columns
specially because its body \vcenter cannot be assigned directly to \adl@box by \setbox15.
Thus we enclose a  $\$ \vcenter{...} \$$  construct in a \hbox and assign it to \adl@box. The
macro \adl@startmbox opens the construct with array’s \@@startpbox, while \adl@endmbox
closes it calling \adl@org@endpbox which is the unmodified \@endpbox of array and mea-
sures the height and depth of the \hbox by \adl@colhtdp.

268 \def\@@startpbox#1{\setbox\adl@box\vtop\bgroup \hsize#1\@arrayparboxrestore}
269 \def\@@endpbox{\unskip \ifhmode \nobreak
270      \vrule\@width\z@\@height\z@\@depth\dp\@arstrutbox \fi
271      \par \egroup \adl@colhtdp \box\adl@box \hfil}
272 \let\@endpbox\@@endpbox
273 \def\adl@startmbox{\bgroup  $\vcenter{\@startpbox}$ }
274 \def\adl@endmbox{\adl@org@endpbox  $\egroup \adl@colhtdp \box\adl@box \hfil}$ 
275
276 %^L

```

## 4.7 Multi-columns

`\multicolumn` The macro `\multicolumn` is modified for the following.

`\adl@preamble`

`\adl@mcaddamp`

`\adl@activatepbox`

- The macros to construct the parts of `\@preamble` for vertical lines, `\adl@arrayrule`, `\adl@arraydashrule` and `\adl@argarraydashrule`, have to perform operations slightly different from those for main preamble. Thus they are `\def`-ined to multi-column version `\adl@mcarrayrule`, etc. These `\def`-initions are enclosed in a group so that they are not affected to `array` or `tabular` which may occur in the third argument of `\multicolumn`. In order to make `\@preamble` work well outside of the group

<sup>15</sup>The author had forgotten this fact until Morten Høgholm pointed out it. Thanks Morten.

containing `\@makepream`, `\adl@preamble` is `\global-ly \let-equal` to `\@preamble` just after `\@makepream` in the group and then reverse `\let`-assignment is performed just after the group is closed. These global assignment is unnecessary with `array` because `\@preamlbe` is constructed `\global-ly`, but safe.

Since this grouping nullifies the effect of `\adl@preamininit` called in `\@mkpream`, we call `\adl@preamininit` again after the group closing.

- In `array`, `\@addamp` to make `\@preamble` for `\multicolumn` has a different definition from that for main one. Thus it is `\let-equal` to `\adl@mcaddamp` whose definition is switched by `\ifadl@usingarypkg`.
- If `array` is in use, `\@preamble` has to be `\xdef`-ed once again by `\@addpreamble` with an `\@empty` argument after `\@mkpreamble` to expand the contents of `\toks` registers. This is performed whether or not with `array` because it is safe.
- As done in `\@array`, `\set@typeset@protect` is replaced with direct `\let`.
- If without `array`, `\@startpbox` and `\@endpbox` should be `\let-equal` to their `@@` counterparts, while should not with `array`. Thus we define `\adl@activatepbox` to do or not to do so depending on `\ifadl@usingarypkg`.
- The counter `\adl@currentcolumn` is `\global-ly` incremented by the first argument of `\multicolumn` (number of columns to be `\span`-ned).

Note that `\adl@columns` is modified by `\@mkpream`, but it is not referred to by `\adl@mcarrayrule` etc., and its value is restored before referred to by `\hdashline`, etc.

```

277
278 %% Multi-Columns
279
280 \def\multicolumn#1#2#3{\multispan{#1}\begingroup \begingroup
281     \def\adl@arrayrule{\adl@mcarrayrule{#1}}%
282     \def\adl@arraydashrule{\adl@mcarraydashrule{#1}}%
283     \def\adl@argarraydashrule{\adl@mcargarraydashrule{#1}}%
284     \let\@addamp\adl@mcaddamp
285     \@mkpream{#2}\@addtopreamble\@empty
286     \global\let\adl@preamble\@preamble \endgroup
287     \let\@preamble\adl@preamble
288     \def\@sharp{#3}\let\protect\relax
289     \adl@activatepbox
290     \adl@preamininit
291     \@arstrut \@preamble\hbox{}\endgroup
292     \global\advance\adl@currentcolumn#1\ignorespaces}
293 \ifadl@usingarypkg
294     \def\adl@mcaddamp{\if@firstamp\@firstampfalse \else\@preamerror5\fi}
295     \let\adl@activatepbox\relax
296 \else
297     \let\adl@mcaddamp\@addamp
298     \def\adl@activatepbox{\let\@startpbox\@@startpbox

```

```

299             \let\@endpbox\@endpbox}
300 \fi
301

```

`\adl@marrayrule` The preamble parts for vertical lines are constructed by the macros `\adl@marrayrule`,  
`\adl@marraydashrule` `\adl@marraydashrule` and `\adl@marraydashrule` to which the first argument  $\langle n \rangle$   
`\adl@marraydashrule` of `\multicolumn` is passed to know the number of columns to be `\span`-ned. They are  
similar to their relatives for main preamble, `\adl@arrayrule`, etc., but the arguments  $\langle c^L \rangle$   
and  $\langle c^R \rangle$  passed to `\adl@xarraydashrule` are;

$$c^L = c, \quad c^R = c + n - 1$$

where  $c = \text{\adl@currentcolumn}$ . This makes leading vertical lines drawn at the left edge of  
the leftmost `\span`-ned column and trailing ones at the right edge of the rightmost column.

```

302 \def\adl@marrayrule#1{\@tempcnta#1\advance\@tempcnta\adl@currentcolumn
303     \advance\@tempcnta\m@ne
304     \adl@xarraydashrule
305         {\adl@currentcolumn}\@tempcnta\{\z@/\z@\}}
306 \def\adl@marraydashrule#1{\@tempcnta#1\advance\@tempcnta\adl@currentcolumn
307     \advance\@tempcnta\m@ne
308     \adl@xarraydashrule
309         {\adl@currentcolumn}\@tempcnta}%
310     {\dashlinedash/\dashlinegap}}
311 \def\adl@marraydashrule#1{\@tempcnta#1\advance\@tempcnta\adl@currentcolumn
312     \advance\@tempcnta\m@ne
313     \adl@xarraydashrule
314         {\adl@currentcolumn}\@tempcnta\{\}%
315     {\chclass\adl@class@iiiorvii \let\@classv\adl@classvfordash}
316
317 %%^L

```

## 4.8 End of Rows

`\@xarraycr` At the end of the  $i^{th}$  row, we have to calculate  $h_i$  which is the height plus depth of the  
`\@xtabularcr` row, and add elements  $\langle C_i^L, h_i \rangle$  and  $\langle C_i^R, h_i \rangle$  to  $R^L$  and  $R^R$ . To do this, `\cr`-s in the  
`\@xargarraycr` macros `\@xarraycr`, `\@xtabularcr`, `\@xargarraycr` are replaced with our own `\adl@cr`.  
`\@yargarraycr` The macro `\@yargarraycr\dimen` is also modified but its `\cr` is replaced with `\adl@`  
`\argcr\dimen` to add (negative) `\dimen` to  $h_i$ . Note that `\@xargarraycr\dimen` uses  
ordinary `\adl@cr` because the extra vertical space of  $\langle \dimen \rangle$  is inserted to the last column.

Note that the implementation of `\@xarraycr` is slightly different between L<sup>A</sup>T<sub>E</sub>X and  
array, we have to have two versions and choose one.

```

318
319 %% End of row
320
321 \ifadl@usingarypkg
322 \def\@xarraycr{\@ifnextchar[\@argarraycr]{\ifnum0='{ }\fi\adl@cr}}
323 \else

```

```

324 \def\@xarraycr{\@ifnextchar[\@argarraycr]{\ifnum0='{ \fi}$\adl@cr}}
325 \fi
326 \def\@xtabularcr{\@ifnextchar[\@argtabularcr]{\ifnum0='{ \fi}\adl@cr}}
327 \def\@xargarraycr#1{\@tempdima#1\advance\@tempdima\dp\@arstrutbox
328     \vrule\@height\z@\@depth\@tempdima\@width\z@
329     \adl@cr}
330 \def\@yargarraycr#1{\adl@argcr{#1}\noalign{\vskip #1}}
331

```

`\adl@cr` The macro `\adl@cr` and `\adl@argcr` perform `\cr` and then invoke the common macro `\adl@cr<x>`. The argument  $\langle x \rangle$  is the extra (negative) vertical space for `\adl@argcr`, while it is 0 for `\adl@cr`.

`\adl@cr` The macro `\adl@cr<x>` at first calculate  $h_i$  as follows. The registers `\adl@height` =  $\eta$  and `\adl@depth` =  $\delta$  have the maximum height and depth of the columns in the row. However, they could be smaller than the height and/or depth of `\@arstrutbox`,  $\eta_s$  and  $\delta_s$ . If so, the height and/or depth of the row are  $\eta_s$  and  $\delta_s$ . Therefore,  $h_i$  is calculated by;

$$h_i = \max(\eta, \eta_s) + \max(\delta, \delta_s).$$

Additionally, if the extra space  $\langle x \rangle$  is negative, a vertical space of  $x$  is inserted below the row<sup>16</sup>. Thus the integer value of  $h_i + x$  is globally added to `\adl@totalheight`, and the elements  $\langle C_i^L = \text{\adl@colsL}, h_i \rangle$  and  $\langle C_i^R = \text{\adl@colsR}, h_i \rangle$  are added to the tail of  $R^L = \text{\adl@rowsL}$  and  $R^C = \text{\adl@rowsR}$ . If  $x$  is not 0 (negative), `discard(x)` or `connect(x)` is also added after  $\langle C_i^{L/R}, h_i \rangle$  according to the current environment (`longtable` or not). In the real implementation,  $R^L$  and  $R^C$  has the following format of  $\langle rows \rangle$ .

$$\begin{aligned}
\langle rows \rangle &::= [\langle row \rangle;]^* \\
\langle row \rangle &::= (\langle cols \rangle / \langle h_i \rangle) \\
\langle cols \rangle &::= [\text{\@elt}\{\langle c \rangle\}\{\langle d \rangle\}\{\langle g \rangle\}]^* \mid \dots C^L \text{ or } C^R \\
&\quad \text{\adl@connect} \mid \dots \text{for } connect(h_i) \\
&\quad \text{\adl@discard} \mid \dots \text{for } discard(h_i) \\
&\quad \text{\relax} \mid \dots \text{for } disconnect(h_i)
\end{aligned}$$

Since `\adl@discard` is defined as `\adl@connect` by `\adl@arrayinit`, added `\adl@discard` transforms itself into `\adl@connect` if current environment is not `longtable`. Otherwise, as we make `\adl@discard` let-equal to `\relax` when a `longtable` environment starts, it keeps its own form.

Then, `\adl@finaldepth` is set to `\adl@depth` if  $x$  is zero, or to zero otherwise (negative), in order to make the depth of `array/tabular` equal to that of the last row. Finally, `\adl@colsL`, `\adl@colsR`, `\adl@currentcolumn`, `\adl@height` and `\adl@depth` are reinitialized to process the next row.

<sup>16</sup>Before v1.54, negative  $\langle x \rangle$  shrinks the height of the row by  $|x|$ . Although the former result may be more appropriate if the row has vertical lines than the current because lines extrude to the next row now, new feature is considered compatible with original `array/tabular`.



```

332 \def\adl@cr{\cr\noalign{\adl@@cr\z@}}
333 \def\adl@argcr#1{\cr\noalign{\adl@@cr{#1}}}}
334 \def\adl@@cr#1{
335     \ifdim\adl@height<\ht\@arstrutbox \adl@height\ht\@arstrutbox\fi
336     \ifdim\adl@depth<\dp\@arstrutbox \adl@depth\dp\@arstrutbox\fi
337     \advance\adl@height\adl@depth
338     \global\advance\adl@totalheight\adl@height
339     \@tempdima#1\relax \global\advance\adl@totalheight\@tempdima
340     \xdef\adl@rowsL{\adl@rowsL
341         (\adl@colsL/\number\adl@height);%
342         \ifdim#1=\z@else (\adl@discard/\number\@tempdima);\fi}%
343     \xdef\adl@rowsR{\adl@rowsR
344         (\adl@colsR/\number\adl@height);%
345         \ifdim#1=\z@else (\adl@discard/\number\@tempdima);\fi}%
346     \gdef\adl@colsL{}\gdef\adl@colsR{}
347     \global\adl@currentcolumn\@ne
348     \ifdim#1=\z@ \global\adl@finaldepth\adl@depth
349     \else \global\adl@finaldepth\z@\fi
350     \global\adl@height\z@ \global\adl@depth\z@}
351
352 %%^L

```

## 4.9 Horizontal Lines

`\hline` The macro `\hline` is modified to insert `\vskip-\arrayrulewidth` before drawing if `\ADLnullwidehline` is in effect, or to add the element  $connect(w) = (\adl@connect/\number\arrayrulewidth)$  to the end of  $R^L$  and  $R^R$  by `\adl@hline` otherwise. The other modifications are to set `\adl@finaldepth` to zero for the case that the last vertical item is `\hline`, and to check if it is followed by not only `\hline` but also `\hdashline` by `\adl@xhline`.

The macro `\cline` is also modified to set `\adl@finaldepth` to zero. As for the feature of `\ADLnullwidehline`, it inserts `\vskip-\arrayrulewidth` to shift the line up before drawing, and `\vskip\arrayrulewidth` after drawing to cancel the negative skip inserted by `\adl@org@cline`.

```

353
354 %% Horizontal Lines
355
356 \def\hline{\noalign{\ifnum0='}\fi
357     \ifadl@zwhrule \vskip-\arrayrulewidth
358     \else \adl@hline\adl@connect\arrayrulewidth \fi
359     \hrule\@height\arrayrulewidth
360     \global\adl@finaldepth\z@
361     \futurelet\@tempa\adl@xhline}
362 \def\cline#1{\noalign{\global\adl@finaldepth\z@
363     \ifadl@zwhrule \vskip-\arrayrulewidth\fi}
364     \adl@org@cline{#1}%
365     \noalign{\ifadl@zwhrule \vskip\arrayrulewidth\fi}}
366

```

`\hdashline` The macro `\hdashline` calls `\adl@hdashline` to open the `\noalign` construct by the well-known trick `{\ifnum0='}\fi` and then to invoke `\adl@ihdashline` checking the existence of its optional argument `[⟨dash⟩/⟨gap⟩]`. Before the invocation, it inserts `\vskip-\arrayrulewidth` if `\ADLnullwidehline` is in effect, or adds `connect(w)` to the end of  $R^L$  and  $R^R$ . Then `\adl@ihdashline` closes the `\noalign` by `\ifnum0='{ \fi}` to start the pseudo row for the horizontal dash-line. Before the dash-line is drawn by `\adl@hcline` which is also used for `\cdashline`, all the columns are `\span`-ned by giving `\adl@columns` to `\multispan`. Finally, the `\noalign` is opened again and `\adl@xhline` is invoked to check whether `\h(dash)line` is followed.

`\adl@inactivehdl` If `\ADLinactivate` is in effect, `\adl@ihdashline` is `\let`-equal to `\adl@inactivehdl`. This macro simply puts a `\hrule` discarding its arguments after inserting `\vskip-\arrayrulewidth` if `\ADLnullwidehline` is in effect.

```

367 \def\hdashline{\adl@hdashline\adl@ihdashline}
368 \def\adl@hdashline#1{\noalign{\ifnum0='}\fi
369     \ifadl@zwhrule \vskip-\arrayrulewidth
370     \else \adl@hline\adl@connect\arrayrulewidth \fi
371     \@ifnextchar[%]
372         {#1}%
373         {#1[\dashlinedash/\dashlinegap]}}
374 \def\adl@ihdashline[#1/#2]{\ifnum0='{ \fi}%
375     \multispan{\adl@columns}\unskip \adl@hcline\z@[#1/#2]%
376     \noalign{\ifnum0='}\fi
377     \futurelet\@tempa\adl@xhline}
378 \def\adl@inactivehdl[#1/#2]{\ifadl@zwhrule \vskip-\arrayrulewidth \fi
379     \hrule\@height\arrayrulewidth
380     \futurelet\@tempa\adl@xhline}
381

```

`\adl@xhline` The macro `\adl@xhline` is the counterpart of the original `\@xhline`. This is introduced to check the mixed sequence of `\hline` and `\hdashline`, and to add the element `disconnect(s) = (\relax/\doublerulesep)` to the end of  $R^L$  and  $R^R$  by `\adl@hline` if a pair of `\h(dash)line` is found.

```

382 \def\adl@xhline{\ifx\@tempa\hline \adl@ixhline\fi
383     \ifx\@tempa\hdashline \adl@ixhline\fi
384     \ifnum0='{ \fi}}
385 \def\adl@ixhline{\vskip\doublerulesep \adl@hline\relax\doublerulesep}
386

```

`\adl@hline` The macro `\adl@hline⟨cs⟩⟨dimen⟩` `\global`-ly adds the integer value of `⟨dimen⟩` to `\adl@totalheight` and adds the element `(⟨cs⟩/\number⟨dimen⟩)` to the tail of  $R^L$  and  $R^R$ . The arguments `⟨cs⟩⟨dimen⟩` are `\adl@connect\arrayrulewidth` for `connect(w)` or `\relax\doublerulesep` for `disconnect(s)`.

```

387 \def\adl@hline#1#2{\@tempcnta#2
388     \global\advance\adl@totalheight\@tempcnta
389     \xdef\adl@rowsL{\adl@rowsL

```

```

390             (#1/\number\@tempcnta);}%
391     \xdef\adl@rowsR{\adl@rowsR
392       (#1/\number\@tempcnta);}}
393

```

`\cdashline` The macro `\cdashline` at first opens `\noalign` and then invokes `\adl@cdline` checking the existence of its optional argument [*dash*]/[*gap*]. The macro `\adl@cdline` first inserts `\adl@cdlinea` `\vskip-\arrayrulewidth` if `\ADLnullwidehline` is in effect. Then it performs column `\adl@cdlineb` `\span-ing` by the code based on that of `\@cline` in L<sup>A</sup>T<sub>E</sub>X-2.09 because L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>'s version will not work with L<sup>A</sup>T<sub>E</sub>X-2.09. The main job is done by `\adl@hcline` after the target columns are `\span`-ned by `\adl@cdlinea` or `\adl@cdlineb`.

`\adl@inactivecdl` If `\ADLinactivate` is in effect, `\adl@cdline` is `\let`-equal to `\adl@inactivecdl`. This macro simply calls our own `\cline`, after closing the `\noalign` opened by `\cdashline`.

```

394 \def\cdashline#1{\noalign{\ifnum0='}\fi
395     \@ifnextchar[%
396       {\adl@cdline[#1]]%
397       {\adl@cdline[#1][\dashlinedash/\dashlinegap]}}
398 \def\adl@cdline[#1-#2]{\ifadl@zwrule \vskip-\arrayrulewidth \fi
399     \global\adl@cla#1\relax
400     \global\advance\adl@cla\m@ne
401     \ifnum\adl@cla>\z@ \global\let\@gtempa\adl@cdlinea
402     \else \global\let\@gtempa\adl@cdlineb\fi
403     \global\adl@clb#2\relax
404     \global\advance\adl@clb-\adl@cla \ifnum0='{\fi}
405     \@gtempa{-\arrayrulewidth}}
406 \def\adl@cdlinea{\multispan\adl@cla &\multispan\adl@clb \unskip \adl@hcline}
407 \def\adl@cdlineb{\multispan\adl@clb \unskip \adl@hcline}
408
409 \def\adl@inactivecdl[#1-#2][#3]{\ifnum0='{\fi}\cline{#1-#2}}
410

```

`\adl@hcline` The macro `\adl@hcline`*w* [*d*]/[*g*] draws a horizontal dash-line of dash size *d* and gap size *g* for `\hdashline` and `\cdashline` in the `\span`-ned columns by `\adl@draw`. As we will discuss in §4.12, the macro requires *d* and *g* are passed through `\@tempdima` and `\@tempdimb`, and control sequences *rule*, *skip* and *box* are passed through its arguments to make it usable for both horizontal and vertical lines. Then the vertical space of *w*, `-\arrayrulewidth` for `\cdashline`, is inserted if it is not 0 (for `\hdashline`) and `\ADLnullwidehline` is not in effect.

```

411 \def\adl@hcline#1[#2/#3]{\@tempdima#2\relax \@tempdimb#3\relax
412     \adl@draw\adl@vrule\hskip\hbox \cr
413     \noalign{\global\adl@finaldepth\z@ \ifdim#1=\z@ \else
414       \ifadl@zwrule \else \vskip#1\fi\fi}}
415

```

`\firsthdashline` If `array` is in use, we wish to have dashed counterparts of `\first/lasthline` named `\first/lasthdashline`, which simply call `\adl@hdashline` with an argument to call `\adl@first/lasthdashline` after closing `\noalign` opened by `\adl@hdashline`.

`\adl@defflhdl`    The macros `\adl@first/lasthdashline`, however, are defined in a tricky manner to replace  
`\adl@idefflhdl`    `\hline` in `\first/lasthline` with;  
`\adl@firsthdashline`  
`\adl@lasthdashline`        `\adl@hdashline\adl@ihdashline[\langle dash \rangle/\langle gap \rangle]`

in order to avoid copy-and-replace. To do that, we define `\adl@defflhdl` and `\adl@idefflhdl` in which the body of `\first/lasthline` is expanded by `\expandafter` and the parts preceding and following `\hline` are extracted. Then the preceding part  $\langle p \rangle$ , the calling sequence of `\adl@hdashline`, and the following part  $\langle f \rangle$  are connected to be the body of `\adl@first/lasthdashline`. Thus we define `\adl@firsthdashline` as follows.

```

\def\adl@firsthdashline[#1/#2]{%
  \langle p \rangle
  \adl@hdashline\adl@ihdashline[#1/#2]
  \langle f \rangle}

416 \ifadl@usingarypkg
417 \def\firsthdashline{\adl@hdashline{\ifnum0='{ \fi}\adl@firsthdashline}}
418 \def\lasthdashline{\adl@hdashline{\ifnum0='{ \fi}\adl@lasthdashline}}
419
420 \def\adl@defflhdl#1{\def\@tempa{#1}
421   \expandafter\adl@idefflhdl}
422 \def\adl@idefflhdl#1\hline#2\@nil{%
423   \@namedef\@tempa[##1/##2]{#1\adl@hdashline\adl@ihdashline[##1/##2]#2}}
424 \adl@defflhdl{\adl@firsthdashline}\firsthline\@nil
425 \adl@defflhdl{\adl@lasthdashline}\lasthline\@nil
426 \fi
427
428 %%^L

```

## 4.10 End of Environment

`\endarray`    The macros to close the `array/tabular` environment, `\endarray` and `\endtabular(*)`,  
`\endtabular`    are modified so that they invoke `\adl@endarray` to draw vertical lines just before clos-  
`\endtabular*`    ing `\halign`, and `\adl@arrayrestore` to restore registers and data structures `\global`-ly  
                  modified in the environment. Note that `array` and related packages such as `delarray` define  
                  a macro `\@arrayright` as the closing hook and thus we invoke it if it is defined.

```

429
430 %% End of Environment
431
432 \def\endarray{\adl@endarray \egroup \adl@arrayrestore \egroup
433   \csname @arrayright\endcsname}
434 \def\endtabular{\endarray $\egroup}
435 \expandafter\let\csname endtabular*\endcsname\endtabular
436

```

`\adl@endarray` The macro `\adl@endarray` at first closes the last row by `\crrc`. If this `\crrc` has real effect, we have to invoke `\adl@cr` to perform our own end-of-row operations. We assume that the `\crrc` is effective if either `\adl@height` or `\adl@depth` has a non-zero value<sup>17</sup>.  
`\adl@rows`  
`\adl@addv1` Then the rows to draw vertical lines  $L_1, \dots, L_n$ ;  
`\adl@vlrowL`  
`\adl@vlrowR`  
`\adl@vlrow`

$$\sigma_1 L_1 \sigma_2 L_2 \dots L_{n-1} \sigma_n L_n \sigma_{n+1}$$

are created in `\adl@vlrowL` and `\adl@vlrowR` by `\adl@makev1rL` and `\adl@makev1rR`. In the real implementation,  $L_k = \langle \gamma_k, \pi_k, \delta_k, \xi_k, \tau_k, \beta_k \rangle$  is represented as;

$$\backslash\mathrm{adl@vl}\{\beta_k\}\{\tau_k - \beta_k\}\{\delta_k\}\{\xi_k\}.$$

Thus `\adl@vl` is made `\let`-equal to `\relax` when the rows are constructed and to `\adl@@vl` when the rows are put.

Since `\adl@makev1rL` and `\adl@makev1rR` shares common macros, they conceptually have the following interface.

```

\adl@vlrow = \adl@makev1rL/R(\adl@rows:\langle R^L \text{ or } R^R \rangle,
                             \adl@currentcolumn:\langle start column \rangle,
                             \adl@addv1:\langle macro to add an element \rangle)

```

Thus they are invoked as;

```

\adl@vlrowL = \adl@makev1rL(\adl@rowsL, 1, \adl@addv1L)
\adl@vlrowR = \adl@makev1rR(\adl@rowsR, \adl@columns, \adl@addv1R)

```

Finally, after constructed rows for vertical lines are put by `\adl@drawv1`, a vertical skip of  $-\mathrm{adl@finaldepth}$  is inserted to move back to the last baseline, and then an invisible `\vrule` of `\adl@finaldepth` deep is put to make `array/tabular` has the depth of the last real row or zero if it ends with a horizontal line.

```

437 \def\adl@endarray{\crrc \noalign{
438     \ifdim\adl@height=\z@
439     \ifdim\adl@depth=\z@ \else \adl@cr\z@ \fi
440     \else \adl@cr\z@ \fi
441     \let\adl@vl\relax
442     \def\adl@vlrow{\adl@currentcolumn\@ne
443         \let\adl@rows\adl@rowsL
444         \let\adl@addv1\adl@addv1L
445         \adl@makev1rL \global\let\adl@vlrowL\adl@vlrow
446     \def\adl@vlrow{\adl@currentcolumn\adl@columns
447         \let\adl@rows\adl@rowsR
448         \let\adl@addv1\adl@addv1R
449         \adl@makev1rR \global\let\adl@vlrowR\adl@vlrow
450     \global\let\adl@vl\adl@@vl}%
451     \adl@drawv1
452     \noalign{\vskip-\adl@finaldepth}%
453     \omit\vrule\@width\z@\@height\z@\@depth\adl@finaldepth\cr}
454

```

<sup>17</sup>The author confesses that this rule is not strict and the introduction of a switch could improve the strictness.

`\adl@arrayrestore` The macro `\adl@arrayrestore` restores the values of registers and data structures, `\adl@height`, `\adl@depth`, `\adl@currentcolumn`, `\adl@totalheight`, `\adl@rowsL`, `\adl@rowsR`, `\adl@colsL` and `\adl@colsR`, saved by `\adl@arrayinit`.

```

455 \def\adl@arrayrestore{%
456     \global\adl@height\adl@heightsave
457     \global\adl@depth\adl@depthsave
458     \global\adl@currentcolumn\adl@currentcolumnsave
459     \global\adl@totalheight\adl@totalheightsave
460     \global\let\adl@rowsL\adl@rowsLsave
461     \global\let\adl@rowsR\adl@rowsRsave
462     \global\let\adl@colsL\adl@colsLsave
463     \global\let\adl@colsR\adl@colsRsave}
464
465 %%^L

```

## 4.11 Drawing Vertical Lines

Figure 2 shows the conceptual code of `\adl@makev1rL`. The correspondance of variables in the code and control sequences in the real implementation is as follows.

$R^L$	: <code>\adl@rowsL</code>	$R$	: <code>\adl@rows</code>	$R'$	: <code>\@tempb</code>	$A$	: <code>\adl@v1rowL</code>
$\Gamma$	: <code>\adl@columns</code>	$\gamma$	: <code>\adl@currentcolumn</code>				
$\tau$	: <code>\@tempcnta</code>	$\beta$	: <code>\@tempcntb</code>	$\eta$	: <code>\adl@lastconn</code>		
$\delta$	: <code>\adl@dash/\adl@dashcolor</code>			$\xi$	: <code>\adl@gap/\adl@gapcolor</code>		
$H$	: <code>\adl@totalheight</code>						
$conn$	: <code>\ifadl@connected</code>			$double$	: <code>\ifadl@doublerule</code>		

`\adl@makev1rL` The macro `\adl@makev1rL` corresponds to the line (2) and (31)–(36). Its right-edge counterpart `\adl@makev1rR` has the same correspondance but the lines (1)–(2) are;

- (1)  $A \leftarrow \langle \rangle$ ;  $R \leftarrow R^R$ ;  $\gamma \leftarrow \Gamma$ ;
- (2) **while**  $\gamma > 0$  **do begin**

and (31)–(36) are;

- (31) **if**  $double$  **then**  $A \leftarrow \langle \backslash\hskip\doublerulesep, A \rangle$ ;
- (32) **else begin**
- (33)  $\gamma \leftarrow \gamma - 1$ ;
- (34) **if**  $\gamma = 0$  **then**  $A \leftarrow \langle \backslash\hss, A \rangle$ ;
- (35) **else**  $A \leftarrow \langle \&\omit\hss, A \rangle$ ;
- (36) **end**;

```

466
467 %% Drawing Vertical Lines
468
469 \def\adl@makev1rL{\adl@makev1r
470     \ifadl@doublerule

```

```

(1)  $\Lambda \leftarrow \langle \rangle$ ;  $R \leftarrow R^L$ ;  $\gamma \leftarrow 1$ ;
(2) while  $\gamma \leq \Gamma$  do begin
(3)    $\tau \leftarrow H$ ;  $\beta \leftarrow H$ ;  $\eta \leftarrow 0$ ;  $\delta \leftarrow \langle -1, \perp \rangle$ ;  $\xi \leftarrow \langle -1, \perp \rangle$ ;
(4)    $conn \leftarrow \text{false}$ ;  $double \leftarrow \text{false}$ ;  $R' \leftarrow \langle \rangle$ 
(5)   while  $R \neq \langle \rangle$  do begin
(6)      $\langle r, R \rangle \leftarrow R$ ;
(7)      $\langle C, h \rangle \leftarrow r$ ;
(8)     if  $C = \langle \rangle$  then begin  $add(\tau, \beta, \delta, \xi)$ ;  $\eta \leftarrow 0$ ; end;
(9)     elseif  $C = \langle connect \rangle$  then  $\eta \leftarrow h$ ;
(10)    else begin
(11)       $\langle e, C' \rangle = C$ ;  $\langle c, d, g \rangle = e$ ;
(12)      if  $c = \gamma$  then begin
(13)        if  $d = \delta \wedge g = \xi$  then begin
(14)          if  $\neg conn$  then begin
(15)             $\tau \leftarrow \beta + \eta$ ;  $conn \leftarrow \text{true}$ ;
(16)          end;
(17)        end;
(18)        else begin
(19)           $add(\tau, \beta, \delta, \xi)$ ;
(20)           $\delta \leftarrow d$ ;  $\xi \leftarrow g$ ;  $\tau \leftarrow \beta + \eta$ ;  $conn \leftarrow \text{true}$ ;
(21)        end;
(22)        if  $C' = \langle \langle \gamma, ?, ? \rangle, ? \rangle$  then  $double \leftarrow \text{true}$ ;
(23)         $C \leftarrow C'$ ;
(24)      end;
(25)      else  $add(\tau, \beta, \delta, \xi)$ ;
(26)       $\eta \leftarrow 0$ ;
(27)    end;
(28)     $\beta \leftarrow \beta - h$ ;  $R' \leftarrow \langle R', \langle C, h \rangle \rangle$ 
(29)  end;
(30)   $add(\tau, \beta, \delta, \xi)$ ;  $R \leftarrow R'$ ;
(31)  if  $double$  then  $\Lambda \leftarrow \langle \Lambda, \backslash \text{hskip} \backslash \text{doublerulesep} \rangle$ ;
(32)  else begin
(33)     $\gamma \leftarrow \gamma + 1$ ;
(34)    if  $\gamma > \Gamma$  then  $\Lambda \leftarrow \langle \Lambda, \backslash \text{hfil} \rangle$ ;
(35)    else  $\Lambda \leftarrow \langle \Lambda, \backslash \text{hfil} \& \backslash \text{omit} \rangle$ ;
(36)  end;
(37) end;
(38)
(39) procedure  $add(\tau, \beta, \delta, \xi)$  begin
(40)   if  $conn$  then begin
(41)      $\Lambda \leftarrow \langle \Lambda, \langle \beta, \tau - \beta, \delta, \xi \rangle \rangle$ ;  $conn \leftarrow \text{false}$ ;
(42)   end;
(43) end;

```

Figure 2: Conceptual Code of `\adl@makev1rL`

```

471         \edef\adl@vlow{\adl@vlow \hskip\doublerulesep}%
472         \let\next\adl@makevrlL
473     \else
474         \advance\adl@currentcolumn\@ne
475         \ifnum\adl@currentcolumn>\adl@columns \let\next\relax
476         \edef\adl@vlow{\adl@vlow \hss}%
477         \else \let\next\adl@makevrlL
478         \edef\adl@vlow{\adl@vlow \hss &\omit}%
479     \fi\fi\next}
480 \def\adl@makevrlR{\adl@makevrl
481     \ifadl@doublerule
482         \edef\adl@vlow{\hskip\doublerulesep \adl@vlow}%
483         \let\next\adl@makevrlR
484     \else
485         \advance\adl@currentcolumn\m@ne
486         \ifnum\adl@currentcolumn=\z@ \let\next\relax
487         \edef\adl@vlow{\hss \adl@vlow}%
488         \else \let\next\adl@makevrlR
489         \edef\adl@vlow{\&\omit \hss \adl@vlow}%
490     \fi\fi\next}
491

```

\adl@makevrl The macro \adl@makevrl corresponds to the lines (3)–(4) and (30).

```

492 \def\adl@makevrl{\@tempcnta\adl@totalheight \@tempcntb\adl@totalheight
493     \let\adl@lastconn\z@ \adl@dash\m@ne \adl@gap\m@ne
494     \let\adl@dashcolor\relax \let\adl@gapcolor\relax
495     \adl@connectedfalse \adl@doublerulefalse \def\@tempb{}%
496     \expandafter\adl@imakevrl\adl@rows\@nil;%
497     \adl@addv1
498     \edef\adl@rows{\@tempb}}
499

```

\adl@imakevrl The macro \adl@imakevrl $\langle r \rangle$ ; corresponds to the lines (5)–(6), and the macro \adl@iimakevrl expands  $\langle C \rangle / \langle h \rangle$  to (7) and (28).

```

\adl@endmakevrl
500 \def\adl@imakevrl#1{\def\@tempa{#1}\ifx\@tempa\@nnil \let\next\relax
501     \else \adl@iimakevrl#1\let\next\adl@imakevrl \fi \next}
502 \def\adl@iimakevrl(#1/#2){\let\@elt\adl@iiimakevrl
503     \def\adl@connect{\adl@connect#2}%
504     \let\adl@endmakevrl\adl@endmakevrlrcut
505     #1\adl@endmakevrl
506     \let\@elt\relax \let\adl@connect\relax
507     \advance\@tempcntb-#2\edef\@tempb{\@tempb(\@tempc/#2);}%
508

```

\adl@iiimakevrl The correspondance of the lines (8)–(30) is a little bit complicated. As shown above, \adl@iimakevrl expands  $C$  attaching the sentinel \adl@endmakevrl.

\adl@vmakevrl  
\adl@endmakevrlrcut  
\adl@endmakevrlconn  
\adl@connect

1. If  $C \neq \langle \rangle$  and  $C \neq \langle connect \rangle$ ,  $C$  has at least one  $\@elt\langle c \rangle\langle d \rangle\langle g \rangle$  which is made \let-equal to \adl@iiimakevrl by \adl@iimakevrl. Thus the lines (10)–(21) and (25)–(26) are performed by \adl@iiimakevrl.



Then;

- (a) if  $c = \gamma$ , `\@elt` becomes `\let-equal` to `\adl@ivmakevlr` which corresponds to (22) in the case of  $C' \neq \langle \rangle$ . Then `\adl@vmakevlr` is invoked for (23) and to eat the sentinel `\adl@endmakevlr`. If  $C' = \langle \rangle$ , `\adl@endmakevlrconn` is invoked, because the sentinel `\adl@endmakevlr` is made `\let-equal` to it by `\adl@iiimakevlr`, for (23) (i.e.  $C \leftarrow \langle \rangle$ ).
  - (b) if  $c \neq \gamma$ , `\adl@vmakevlr` is invoked to perform implicit  $C \leftarrow C$  operation and to eat the sentinel.
2. If  $C = \langle connect \rangle$ , i.e. it has only one element `\adl@connect`, the macro `\adl@@connect` is invoked with  $h$  because it is `\define-dl` to be `\adl@@connect\langle h \rangle`. The macro performs (9) and implicit  $C \leftarrow C (= \langle connect \rangle)$  eating the sentinel.
  3. If  $C = \langle \rangle$ , `\adl@endmakevlrcut` that is `\let-equal` to the sentinel `\adl@endmakevlr` is invoked to perform (8) and implicit  $C \leftarrow C (= \langle \rangle)$ .

```

509 \def\adl@iiimakevlr#1#2#3#4#5{\let\@elt\adl@ivmakevlr \let\next\relax
510     \ifnum#1=\adl@currentcolumn\relax
511         \let\adl@endmakevlr\adl@endmakevlrconn
512         \@tempswafalse
513         \ifnum#2=\adl@dash\relax
514         \ifnum#3=\adl@gap\relax
515         \def\@tempa{#4}\ifx\@tempa\adl@dashcolor
516         \def\@tempa{#5}\ifx\@tempa\adl@gapcolor
517             \@tempswatrue
518         \fi\fi\fi\fi
519         \if@tempswa
520             \ifadl@connected\else
521                 \@tempcnta\@tempcntb
522                 \advance\@tempcnta\adl@lastconn\relax
523                 \adl@connectedtrue
524             \fi
525         \else
526             \adl@addv1
527             \adl@dash#2\relax \adl@gap#3\relax
528             \def\adl@dashcolor{#4}\def\adl@gapcolor{#5}%
529             \@tempcnta\@tempcntb
530             \advance\@tempcnta\adl@lastconn\relax
531             \adl@connectedtrue
532         \fi
533     \else
534         \adl@addv1
535         \def\next{\adl@vmakevlr\@elt{#1}{#2}{#3}{#4}{#5}}%
536     \fi
537     \let\adl@lastconn\z@ \next}
538 \def\adl@ivmakevlr#1{%
539     \ifnum#1=\adl@currentcolumn \adl@doubleruletrue \fi
540     \adl@vmakevlr\@elt{#1}}

```

```

541 \def\adl@vmakev1r#1\adl@endmakev1r{\def\@tempc{#1}}
542 \def\adl@endmakev1rcut{\adl@adv1 \let\adl@lastconn\z@ \def\@tempc{}}
543 \def\adl@endmakev1rconn{\def\@tempc{}}
544 \def\adl@@connect#1\adl@endmakev1r{\def\adl@lastconn{#1}%
545     \def\@tempc{\adl@connect}}
546

```

\adl@adv1L The macro \adl@adv1L corresponds to the lines (38)–(42), i.e. the procedure *add*. The  
\adl@adv1R macro \adl@adv1R performs similar operations, but its conceptual code is the following.

```

(38) procedure add( $\tau, \beta, \delta, \xi$ ) begin
(39)     if conn then begin
(40)          $\Lambda \leftarrow \langle \beta, \tau - \beta, \delta, \xi \rangle, \Lambda$ ; conn  $\leftarrow$  false;
(41)     end;
(42) end;

```

```

547 \def\adl@adv1L{\ifadl@connected
548     \advance\@tempcnta-\@tempcntb
549     \edef\adl@v1row{\adl@v1row
550         \adl@v1{\number\@tempcntb}{\number\@tempcnta}%
551         {\number\adl@dash}{\number\adl@gap}%
552         {\adl@dashcolor}{\adl@gapcolor}}%
553     \adl@connectedfalse \fi}
554 \def\adl@adv1R{\ifadl@connected
555     \advance\@tempcnta-\@tempcntb
556     \edef\adl@v1row{\adl@v1{\number\@tempcntb}{\number\@tempcnta}%
557         {\number\adl@dash}{\number\adl@gap}%
558         {\adl@dashcolor}{\adl@gapcolor}\adl@v1row}%
559     \adl@connectedfalse \fi}
560

```

\adl@drawv1 After the the macros \adl@v1rowL and \adl@v1rowR are constructed, they are expanded to  
\adl@@v1 draw vertical lines by \adl@drawv1. Prior to the expansion, the macro \adl@drawv1 glob-  
\adl@v1@leftskip ally defines \adl@v1@leftskip and \adl@v1@rightskip, which are the amount of negative  
\adl@v1@rightskip spaces inserted to the left/right of a vertical line, as follows.

$$\begin{aligned}
 \adl@v1@leftskip &= \begin{cases} \arrayrulewidth/2 & \text{if } \ifadl@zwrule \\ 0 & \text{else if leftside} \\ \arrayrulewidth & \text{otherwise} \end{cases} \\
 \adl@v1@rightskip &= \begin{cases} \arrayrulewidth/2 & \text{if } \ifadl@zwrule \\ 0 & \text{else if rightside} \\ \arrayrulewidth & \text{otherwise} \end{cases}
 \end{aligned}$$

That is, if \ADLnulwide is in effect, a vertical line is surrounded by horizontal spaces of  $-\arrayrulewidth/2$  to adjust the center of the line to the left or right edge of its column. Otherwise, a horizontal space  $-\arrayrulewidth$  is inserted after (before) the line is drawn to adjust its left (right) edge to the left (right) edge of the column<sup>18</sup>.

<sup>18</sup>Before v1.54, the horizontal spaces was not inserted if \ADLsomewide and thus disconnected lines were not aligned vertically.

Then the macros `\adl@vrowL` and `\adl@vrowR` are expanded. These macros will have `\adl@v1`, which is made `\let=equal` to `\adl@@v1` prior to the expansion, to draw a vertical line. The macro `\adl@@v1<β><λ><δllcc ( $x_l$  and  $x_c$  are length and color) draws a solid line if  $\gamma_l = 0$  or a dash-line otherwise in a \vbox of  $\lambda = \tau - \beta$  high and \raise-s it by  $\beta$ . The method to draw a dash line in the \vbox is analogous to that for horizontal line shown in §4.9, except that a line is surrounded by horizontal spaces of \adl@v1@leftskip and \adl@v1@rightskip. Coloring gaps is done by drawing a vertical rule setting  $\gamma_c$  by \set@color prior to dash line drawing if  $\gamma_c$  is not \relax. To color dashes or solid line, \set@color with  $\delta_c$  is done if it is not \relax before line drawing.`

```

561 \def\adl@drawv1{%
562     \omit \relax \ifadl@zwvrule
563         \gdef\adl@v1@leftskip{.5\arrayrulewidth}%
564         \global\let\adl@v1@rightskip\adl@v1@leftskip
565     \else
566         \global\let\adl@v1@leftskip\z@
567         \global\let\adl@v1@rightskip\arrayrulewidth
568     \fi \adl@vrowL \cr
569 \omit \relax \ifadl@zwvrule
570     \gdef\adl@v1@leftskip{.5\arrayrulewidth}%
571     \global\let\adl@v1@rightskip\adl@v1@leftskip
572 \else
573     \global\let\adl@v1@leftskip\arrayrulewidth
574     \global\let\adl@v1@rightskip\z@
575 \fi \adl@vrowR \cr}
576
577 \def\adl@@v1#1#2#3#4#5#6{\vbox to\z@{\vss\hbox{%
578     \hskip-\adl@v1@leftskip
579     \ifnum#3=\z@ \else \def\@tempa{#6}\ifx\@tempa\adl@nocolor \else
580         \raise#1sp\hbox{\let\current@color\@tempa \set@color
581             \vrule height#2sp width\arrayrulewidth}%
582         \hskip-\arrayrulewidth \fi \fi
583     \raise#1sp\vbox to#2sp{
584         \def\@tempa{#5}\ifx\@tempa\adl@nocolor \else
585             \let\current@color\@tempa \set@color \fi
586         \ifnum#3=\z@
587             \hrule height#2sp depth\z@ width\arrayrulewidth
588         \else
589             \@tempdima#3sp \@tempdimb#4sp
590             \adl@draw\adl@hrule\vskip\vbox
591         \fi}%
592     \hskip-\adl@v1@rightskip}}
593
594 %%^L

```

## 4.12 Drawing Dash-lines

`\adl@vrule` As explained later, horizontal and vertical lines are drawn by a common macro `\adl@draw`  
`\adl@hrule` to which the length of a dash segment,  $d$ , is passed through `\@tempdima`. The macro also has an argument that is either `\adl@vrule` to draw a dash for *horizontal* lines or `\adl@hrule` for *vertical*. These two macros commonly have one argument  $\langle f \rangle$  to draw a dash of  $f \times d$  long and of `\arrayrulewidth` wide.

```

592
593 %% Draw Dash Lines (\adl@vrule/\adl@hrule, \hskip/\vskip, \hbox/\vbox)
594
595 \def\adl@vrule#1{\vrule\@width#1\@tempdima\@height\arrayrulewidth\relax}
596 \def\adl@hrule#1{\hrule\@height#1\@tempdima\@width\arrayrulewidth\relax}

```

`\adl@draw` The macro `\adl@draw` is to draw a horizontal or vertical line. It is `\let`-equal to one of `\adl@drawi`, `\adl@drawii` and `\adl@drawiii` according to the drawing mode specified by `\ADLdrawingmode`. These three macros have common interface, `\@tempdima` and `\@tempdimb` for the length of dash and gap,  $d$  and  $g$ , and three arguments  $\langle rule \rangle$ ,  $\langle skip \rangle$  and  $\langle box \rangle$  with which `\adl@draw` is called in the following manner.

```

\adl@draw\adl@vrule\hskip\hbox ... horizontal
\adl@draw\adl@hrule\vskip\vbox ... vertical

```

The drawing methods in three modes have been explained in §4.2. More specifically, `\adl@drawi` for mode 1, to which `\adl@draw` is `\let`-equal by default, conceptually performs the following operations.

$$\begin{aligned}
 & \langle rule \rangle \{1/2\} \quad \langle skip \rangle (g/2) \\
 & \backslash xleaders \langle box \rangle \{ \langle skip \rangle (g/2) \quad \langle rule \rangle \{1\} \quad \langle skip \rangle (g/2) \} \\
 & \quad \langle skip \rangle (0 \text{ plus } lfil \text{ minus } lfil) \\
 & \langle skip \rangle (g/2) \quad \langle rule \rangle \{1/2\}
 \end{aligned}$$

The conceptual operations of `\adl@drawii` for mode 2 are as follows.

$$\begin{aligned}
 & \langle rule \rangle \{1/2\} \quad \langle skip \rangle (g/2) \\
 & \langle box \rangle \{ \langle skip \rangle (g/2) \quad \langle rule \rangle \{1\} \quad \langle skip \rangle (g/2) \} \quad \langle skip \rangle (-d - g) \\
 & \backslash xleaders \langle box \rangle \{ \langle skip \rangle (g/2) \quad \langle rule \rangle \{1\} \quad \langle skip \rangle (g/2) \} \\
 & \quad \langle skip \rangle (0 \text{ plus } lfil \text{ minus } lfil) \\
 & \langle skip \rangle (-d - g) \quad \langle box \rangle \{ \langle skip \rangle (g/2) \quad \langle rule \rangle \{1\} \quad \langle skip \rangle (g/2) \} \\
 & \langle skip \rangle (g/2) \quad \langle rule \rangle \{1/2\}
 \end{aligned}$$

The macro `\adl@drawiii` for mode 3 is quite similar to `\adl@drawi` except that `\xleaders` is replaced by `\cleaders`. This replacement is done by temporarily `\let`-ing `\xleaders` be equal to `\cleaders`.

```

597 \def\adl@drawi#1#2#3{%
598     #1{.5}#2.5\@tempdimb
599     \xleaders#3{#2.5\@tempdimb #1{1}#2.5\@tempdimb}%
600     #2\z@ plus1fil minus1fil\relax
601     #2.5\@tempdimb #1{.5}}
602 \def\adl@drawii#1#2#3{%
603     \setbox\adl@box#3{#2.5\@tempdimb #1{1}#2.5\@tempdimb}%
604     #1{.5}#2.5\@tempdimb
605     \copy\adl@box #2-\@tempdima #2-\@tempdimb
606     \xleaders\copy\adl@box#2\z@ plus1fil minus1fil\relax
607     #2-\@tempdima #2-\@tempdimb \copy\adl@box
608     #2.5\@tempdimb #1{.5}}

```

```

609 \def\adl@drawiii#1#2#3{{\let\xleaders\cleaders \adl@drawi#1#2#3}}
610 \let\adl@draw\adl@drawi
611

```

`\ADLdrawingmode` The macro `\ADLdrawingmode{<m>}` defines the drawing mode by `\let`-ing `\adl@draw` be equal to `\adl@drawi` if  $m = 1$ , and so on. If  $\langle m \rangle$  is neither 1, 2 nor 3, it is assumed as 1.

```

612 \def\ADLdrawingmode#1{\ifcase #1%
613     \let\adl@draw\adl@drawi \or
614     \let\adl@draw\adl@drawi \or
615     \let\adl@draw\adl@drawii \or
616     \let\adl@draw\adl@drawiii \else
617     \let\adl@draw\adl@drawi \fi}
618
619 %%^L

```

### 4.13 Shorthand Activation

`\adl@Array` The macros `\adl@Array`, `\adl@Tabular`, `\adl@Tabular*` and `\adl@Longtable` start environments `array`, `tabular`, `tabular*` and `longtable` respectively, turning `\ifadl@`  
`\adl@Tabular` `array`, `tabular`, `tabular*` and `longtable` respectively, turning `\ifadl@`  
`\adl@Tabularstar` `inactive` false to activate dash-line functions. We will `\let` macros `\Array` etc. be equal  
`\adl@Longtable` to them for shorthand activation.

```

620
621 %% Shorthand Activation
622
623 \def\adl@Array{\adl@inactivefalse \array}
624 \def\adl@Tabular{\adl@inactivefalse \tabular}
625 \def\adl@Tabularstar{\adl@inactivefalse \@nameuse{tabular*}}
626 \def\adl@Longtable{\adl@inactivefalse \longtable}
627

```

`\@notdefinable` Before making `\Array` etc. `\let`-equal to `\adl@Array` etc., we have to check if these macros  
`\adl@notdefinable` having too natural names have already used. This check is done by `\@ifdefinable` that will call `\@notdefinable` for the complaint if undefinable. Since we want to complain with our own warning message, `\@notdefinable` is temporarily `\def`-ined so that it simply `\def`-ines a macro `\adl@notdefinable` as empty. Therefore, `\adl@notdefinebale` will have some definition if one of `\Array`, `\Tabular`, `\Tabular*` and `\Longtable` (if `longtable` is loaded) cannot be defined, while it will stay undefined otherwise.

```

628 \begingroup
629 \def\@notdefinable{\gdef\adl@notdefinable{}}
630 \@ifdefinable\Array\relax
631 \@ifdefinable\Tabular\relax
632 \expandafter\@ifdefinable\csname Tabular*\endcsname\relax
633 \ifx\longtable\undefined\else \@ifdefinable\Longtable\relax \fi
634 \endgroup
635

```

`\Array` If `\adl@notdefinable` is `\undefined` indicating that all `\Array` etc. are definable, we `\let`  
`\Tabular` them be equal to `\adl@Array` etc. We also `\let` ending macros `\endArray` etc. be equal to  
`\Tabular*` `\endarray` etc. Note that `\Longtable` and `\endLongtable` are defined only when `longtable`  
`\Longtable` is loaded, and `\endLongtable` is `\def`-ined as (not being `\let`-equal to) `\endlongtable`  
`\endArray` because its definition of our own is not given yet.  
`\endTabular` Otherwise, we complain with a warning message put by `\PackageWarning` if it is defined  
`\endTabular*` (i.e.  $\text{\LaTeX 2}_{\epsilon}$ ) or `\@warning` otherwise (i.e.  $\text{\LaTeX-2.09}$ ).  
`\endLongtable`

```

636 \ifx\adl@notdefinable\undefined
637     \let\Array\adl@Array
638     \let\Tabular\adl@Tabular
639     \expandafter\let\csname Tabular*\endcsname\adl@Tabularstar
640     \let\endArray\endarray
641     \let\endTabular\endtabular
642     \expandafter\let\csname endTabular*\endcsname\endtabular
643     \ifx\longtable\undefined\else
644         \let\Longtable\adl@Longtable
645         \def\endLongtable{\endlongtable}
646     \fi
647 \else
648 \begingroup
649 \ifx\longtable\undefined
650 \def\@tempa{Array and Tabular are not defined because one of them\MessageBreak
651     has been defined}
652 \else
653 \def\@tempa{Array/Tabular/Longtable are not defined because \MessageBreak
654     one of them has been defined}
655 \fi
656 \ifx\PackageWarning\undefined
657     \def\MessageBreak{^^J}
658     \@warning\@tempa
659 \else
660     \let\on@line\empty
661     \PackageWarning{arydshln}\@tempa
662 \fi
663 \endgroup
664 \fi
665

```

`\ADLnoshorthanded` If a user wishes to define an environment named `Array` or `Tabular(*)` (or `Longtable` if `longtable` is in use) by him/herself or by loading other packages *after* `arydshln` is loaded, `\newenvironment` for `Array` etc. will fail because they have already been undefinable. The macro `\ADLnoshorthanded` makes them definable again by `\let`-ing them and their ending counterparts be equal to `\relax`.

```

666 \def\ADLnoshorthanded{%
667     \let\Array\relax
668     \let\Tabular\relax
669     \expandafter\let\csname Tabular*\endcsname\relax
670     \let\endArray\relax

```

```

671      \let\endTabular\relax
672      \expandafter\let\csname endTabular*\endcsname\relax
673      \ifx\longtable\undefined\else
674          \let\Longtable\relax
675          \let\endLongtable\relax \fi}
676

```

\adl@act@arrayclassz Finally here we define *active* version of \@arrayclassz named \adl@act@arrayclassz etc. for \adl@activate (see §4.4). The definitions are simply done by \let-ing \adl@act@arrayclassz equal to \@arrayclassz etc<sup>19</sup>.

```

\adl@act@@startpbox
\adl@act@@endpbox 677 \let\adl@act@arrayclassz\@arrayclassz
\adl@act@endpbox 678 \let\adl@act@tabclassz\@tabclassz
\adl@act@cr 679 \ifadl@usingarypkg \let\adl@act@classz\@classz \fi
\adl@act@argcr 680 \let\adl@act@@startpbox\@startpbox
\adl@act@ccline 681 \let\adl@act@@endpbox\@endpbox
\adl@act@endarray 682 \let\adl@act@endpbox\@endpbox
\adl@act@hline 683 \let\adl@act@cr\adl@cr
\adl@act@ihdashline 684 \let\adl@act@argcr\adl@argcr
\adl@act@cdline 685 \let\adl@act@endarray\adl@endarray
\adl@act@@vlineL 686 \let\adl@act@hline\adl@hline
\adl@act@@vlineR 687 \let\adl@act@ihdashline\adl@ihdashline
688 \let\adl@act@cdline\adl@cdline
689 \let\adl@act@@vlineL\adl@@vlineL
690 \let\adl@act@@vlineR\adl@@vlineR
691
692 %%^L

```

#### 4.14 Compatibility with colortab

\adl@CC@ The package colortab has a macro;  
\CC@

\LCC<colorspec>\<rows>\ECC

to color <rows> referring <colorspec>. The macro \CC@, the heart of the coloring function, first makes a box with <rows> using \@preamble to measure the height of <rows>, then makes a row putting a heavy rule of the height in each column with a color command for the column specified by <colorspec>, and finally puts <rows> overlaying them on the colored rule. Therefore <rows> is processed twice by \CC@ to update \global registers/structures incorrectly.

Thus we modify \CC@, if the package colortab is provided, to save \global stuff by \adl@arraysave before the height measurement and restore them by \adl@arrayrestore after that.

```

693
694 %% Compatibility with colortab
695

```

---

<sup>19</sup>Alternatively, we may define \adl@act@arrayclassz in place of \@arrayclassz but the author chose this way to minimize the possibility of *enbug*.

```

696 \def\adl@CC@#1#2#3{%
697   \ifcolortab
698     \noalign{%
699       \adl@arraysave
700       \setbox\CT@box=\vbox{#1#3\crr\egroup}%
701       \adl@arrayrestore
702       \CT@dim=\ht\CT@box
703       \global\advance\CT@dim by \dp\CT@box
704       \def\CT@next{}%
705       \futurelet\next\CT@columncolor#2&\@nil}%
706     \CT@next\cr
707     \noalign{\vskip-\CT@dim}%
708   \fi
709   #3}
710 \ifx\ColortabLoaded\undefined\else
711 \let\CC@\adl@CC@
712 \fi
713
714 %%^L

```

## 4.15 Compatibility with longtable

Making `arydshln` compatible with `longtable` is a hard job because a `longtable` consists of multiple *chunks* and each chunk is a distinct `\halign`. We could draw vertical lines in each chunk as we do with ordinary `array/table`. However this straightforward solution should *break* dash-lines at invisible borders of chunks and produce awful results.

Therefore, this implementation draws dash-lines in `\output` routine in which we have all the rows to be put in a page. The hard part is to know which rows are being put in `\output`. This problem is solved by extracting the leading part of  $R^L$  (`\adl@rowsL`) and  $R^R$  (`\adl@rowsR`) by the height/depth of the table fraction to be put and removing the part from  $R^{L/R}$ .

### 4.15.1 Initialization

First of all, the following switch and `\dimen` register are declared.

- |                                 |  |
|---------------------------------|--|
| <code>\ifadl@LTfirstpage</code> | <ul style="list-style-type: none"> <li><code>\ifadl@LTfirstpage</code> is tested in <code>\output</code> routine to examine if the page being put has the first fraction of a <code>longtable</code>.</li> </ul>   |
| <code>\adl@LTpagetotal</code>   | <ul style="list-style-type: none"> <li><code>\adl@LTpagetotal</code> is set to <code>\pagetotal</code> just before the first portion of a <code>longtable</code> is added to the main vertical list. Since the <code>\box255</code> has items preceding the <code>\longtable</code> and its first fraction, we can obtain the height of the first fraction by subtracting <code>\adl@LTpagetotal</code> from the height plus depth of <code>\box255</code>.</li> </ul> |

```

715
716 %% Compatibility with longtable: initialization
717
718 \newif\ifadl@LTfirstpage
719 \newdimen\adl@LTpagetotal
720

```



Next, we skip everything if `longtable` is not in use, or we have undefined-error when we refer to the definitions in it. Note that since `\newif` cannot be in the `\ifx/\fi` construct, the declarations above are excluded.

```
721 \ifx\longtable\undefined\else
722
```

`\adl@LT@array` Then we redefine the macro `\LT@array`, which is the heart of `\longtable`, saving its original definition in `\adl@LT@array`. The modified `\LT@array` first calls `\adl@arrayinit` to initialize the global data structures, and sets `\ifadl@LTfirstpage` to true. Then `\adl@dashline`, `\adl@idashline` and `\adl@discard` are made `\let`-equal to the `longtable` versions `\adl@LTdashline` and `\adl@LTidashline`, and `\relax` (to inhibit expansion) respectively. Then the macro calls `\adl@LTinactivate` if `\adl@inactive` is true, and finally calls its original version `\adl@LT@array`. Note that since `longtable` cannot be nested;

- `\adl@arraysave` in `\adl@arrayinit` is unnecessary but safe, and thus its invocation timing is not so sensitive; and
- activator is not required.

Also note that the assignment `\adl@ncol` to `\adl@columns` in `\adl@arrayinit` is void and thus we will do it afterward.

`\adl@LTinactivate` The macro `\adl@LTinactivate` first calls `\adl@inactivate` to do basic inactivation and then `\let`-s the following control sequences be equal to their counterparts in `longtable`.

```
\endlongtable \LT@make@row \LT@echunk \LT@end@hd@ft \LT@kill
\LT@output
```

It also make `\adl@idashline` `\let`-equal to its inactive version because we need the macro to find mixed `\hline` and `\hdashline` sequence.

```
723 \let\adl@LT@array\LT@array
724 \def\LT@array{\adl@arrayinit \adl@LTfirstpagetrue
725     \let\adl@discard\relax \let\adl@hdashline\adl@LTdashline
726     \let\adl@ihdashline\adl@LTidashline
727     \ifadl@inactive \adl@LTinactivate \fi
728     \adl@LT@array}
729 \def\adl@LTinactivate{\adl@inactivate
730     \let\endlongtable\adl@org@endlongtable
731     \let\LT@make@row\adl@org@LT@make@row
732     \let\LT@echunk\adl@org@LT@echunk
733     \let\LT@end@hd@ft\adl@org@LT@end@hd@ft
734     \let\LT@kill\adl@org@LT@kill
735     \let\LT@output\adl@org@LT@output
736     \let\adl@ihdashline\adl@LTinactivehdl}
737
```

`\adl@org@LT@make@row` The macro `\LT@make@row` is redefined for additional initialization which must be done after the original `\LT@array` performs its own initialization. First, `\LT@make@row` itself is reset to its original version `\adl@org@LT@make@row` to initialize stuff only once, since `\LT@make@`

`row` is called repeatedly at each chunk. Next `\adl@ncol` is assigned to `\adl@columns` to give its value calculated in `\@mkpream`. Then macros to begin/end p-boxes are made `\let`-equal to our own version because the original `\LT@array` has done it with its own version. Note that `\@@startpbox` and `\@statpbox` are `\let`-equal to our own `\adl@LTstartpbox` if `array` is not in use because with `array` opening a p-box is not done by `\@startpbox` but is embedded in `\@preamble`. Also note that we need `\adl@LTendmbox` to close m-boxes through our own closing macro `\adl@endmbox`, whose definition is kept in `\adl@endmbox`, for `longtable`-specific operations for footnotes. Finally, the original version `\adl@org@LT@make@row` is called.

```

738 \let\adl@org@LT@make@row\LT@make@row
739 \def\LT@make@row{\let\LT@make@row\adl@org@LT@make@row
740     \adl@columns\adl@ncol
741     \ifadl@usingarypkg\else
742         \let\@@startpbox\adl@LTstartpbox
743         \let\@startpbox\adl@LTstartpbox \fi
744     \let\@@endpbox\adl@LTendpbox
745     \let\@endpbox\adl@LTendpbox
746     \let\adl@endmbox\adl@endmbox
747     \let\adl@endmbox\adl@LTendmbox
748     \adl@org@LT@make@row}
749
750 %%^L

```

The summary of the activation and inactivation specific to `longtable` is shown in Table 2.

#### 4.15.2 Ending Chunks

<pre> \adl@org@endlongtable \endlongtable \adl@org@LT@echunk \LT@echunk \adl@LTlastrow </pre>	<p>When a chunk is closed with <code>\crrc</code>, we have to add the information of the last row to <math>R^{L/R} = \text{\adl@rows}L/R</math> if the row is not finished by an explicit <code>\.</code>. This is done by <code>\adl@LTlastrow</code> as we did at the first job of <code>\adl@endarray</code>. Two chunk closing macros, <code>\endlongtable</code> and <code>\LT@echunk</code>, are modified to call <code>\adl@LTlastrow</code> before its original job done by <code>\adl@org@endlongtable</code> and <code>\adl@org@LT@echunk</code> respectively. Note that <code>\adl@LTlastrow</code> only has <code>\crrc</code> and <code>\noalign</code> and thus another <code>\crrc</code> in original <code>\endlongtable</code> and <code>\LT@echunk</code> is no-operation as desired. Also note that <code>\adl@LTlastrow</code> is called twice from <code>\endlongtable</code>, once from <code>\LT@echunk</code> in the original version, but it is safe because the first call makes <code>\adl@height</code> and <code>\adl@depth</code> zero and thus the second become no-operation.</p>
---	---

```

751
752 %% Compatibility with longtable: end chunk
753
754 \let\adl@org@endlongtable\endlongtable
755 \def\endlongtable{\adl@LTlastrow \adl@org@endlongtable}
756
757 \let\adl@org@LT@echunk\LT@echunk
758 \def\LT@echunk{\adl@LTlastrow \adl@org@LT@echunk}
759
760 \def\adl@LTlastrow{\crrc \noalign{

```

Table 2: Active and Inactive longtable Operations

command	active	inactive
p b (open) with array	\adl@act@classz →\LT@startpbox	\adl@org@classz →\LT@startpbox
without array	\adl@LTstartpbox	\LT@startpbox
m (open)	\adl@act@classz →\adl@startmbox →\LT@startpbox	\adl@org@classz →\LT@startpbox
p b (close)	\adl@LTendpbox	\LT@endpbox
m (close)	\adl@LTendmbox	\LT@endpbox
\hline	→\adl@act@hline	→\@gobbletwo
\hdashline	→\adl@LTihdashline →\adl@act@hline	→\adl@LTinactivehdl →\@gobbletwo
\endlongtable \LT@make@row \LT@echunk \LT@end@hd@ft \LT@kill \LT@output	modified version	\adl@org@endlongtable \adl@org@LT@make@row \adl@org@LT@echunk \adl@org@LT@end@hd@ft \adl@org@LT@kill \adl@org@LT@output

```

761      \ifdim\adl@height=\z@
762      \ifdim\adl@depth=\z@   \else \adl@@cr\z@ \fi
763                          \else \adl@@cr\z@ \fi}}
764

```

Another chunk ending macro is \LT@end@hd@ft{*box*} to close a header/footer called by \LT@end@hd@ft, \endfirsthead, \endhead, \endlastfoot and \endfoot with an argument *box* being \LT@firsthead, \LT@head, \LT@lastfoot and \LT@foot respectively. In order to maintain the information of rows  $R^{L/R} = \text{\adl@rowsL/R}$  of headers/footers separately from the main one, the modified \LT@end@hd@ft saves them together with \adl@totalheight to weirdly named macros;

```

\adl@org@LT@end@hd@ft
\LT@end@hd@ft
\adl@LT@hfsave
\adl@LT@th
\\adl@LT@th\LT@firsthead
\\adl@LT@th\LT@head
\\adl@LT@th\LT@lastfoot
\\adl@LT@th\LT@foot
\\adl@rowsL\LT@firsthead
\\adl@rowsL\LT@head
\\adl@rowsL\LT@lastfoot
\\adl@rowsL\LT@foot
\\adl@rowsR\LT@firsthead
\\adl@rowsR\LT@head
\\adl@rowsR\LT@lastfoot
\\adl@rowsR\LT@foot

```

after closing the last row by \adl@LT@lastrow. The \string representation of the macros looks like;

```
\\adl@LT@th\LT@firsthead
```

and so on. The saving operation is done by the macro \adl@LT@hfsave{*box*}{*info*} and is equivalent to;

```
\global\let\<info>\<box>=\<info>
```

After the saving, three global variables are reinitialized. Calling `\adl@LTlastrow` twice, once from the original version through `\LT@echunk` is safe as described above.

```

765 \let\adl@org@LT@end@hd@ft\LT@end@hd@ft
766 \def\LT@end@hd@ft#1{\adl@LTlastrow
767     \noalign{\edef\adl@LTth{\number\adl@totalheight}%
768         \adl@LTthsave#1\adl@LTth \global\adl@totalheight\z@
769         \adl@LTthsave#1\adl@rowsL\gdef\adl@rowsL{}%
770         \adl@LTthsave#1\adl@rowsR\gdef\adl@rowsR{}}
771     \adl@org@LT@end@hd@ft#1}
772 \def\adl@LTthsave#1#2{\expandafter\global\expandafter\let
773     \csname\string#2\string#1\endcsname#2}
774

```

`\adl@org@LT@kill` The additional job for yet another chunk closer `\LT@kill` to kill a template row is a little bit harder. Since the row information might have been added by an explicit `\kill` preceding `\adl@LTkill`, we have to remove it from the tail of `\adl@rowsL/R`, and subtract its  $h_i$  from `\adl@totalheight` because `\kill`-ed row may be in header/footer definition. To do that, modified `\LT@kill` first ensures the information addition by `\adl@LTlastrow`, then traverses `\adl@rowsL/R` adding its non-last elements to `\@tempb` by the loop of `\adl@LTkill`, and assigns `\@tempb` to `\adl@rowsL/R` globally by `\adl@LTkillend` when `\adl@LTkill` finds the tail. The macro `\adl@LTkillend` also sets the  $h_i$  of the last element to `\@tempcnta`, which is subtracted from `\adl@totalheight` globally. Finally, the original version `\adl@org@LT@kill` is called.

```

775 \let\adl@org@LT@kill\LT@kill
776 \def\LT@kill{\adl@LTlastrow \noalign{
777     \def\@tempb{}\expandafter\adl@LTkill\adl@rowsL\@nil\adl@rowsL
778     \def\@tempb{}\expandafter\adl@LTkill\adl@rowsR\@nil\adl@rowsR
779     \global\advance\adl@totalheight-\@tempcnta}%
780     \adl@org@LT@kill}
781 \def\adl@LTkill#1;#2{\def\@tempa{#2}%
782     \ifx\@tempa\@nnil\def\next{\adl@LTkillend#1}%
783     \else\edef\@tempb{\@tempb#1;}\def\next{\adl@LTkill#2}\fi
784     \next}
785 \def\adl@LTkillend(#1/#2)#3{\global\let#3\@tempb \@tempcnta#2\relax}
786
787 %%^L

```

### 4.15.3 Horizontal Lines and p-Boxes

`\LT@hline` The macro `\LT@hline`, longtable version of `\hline`, is redefined to add pseudo row information to  $R^{L/R}$  and to check mixed sequence of `\hline` and `\hdashline`<sup>20</sup>. We also define the macro `\adl@LTihdashline` [*dash*]/[*gap*] and its inactive counterpart `\adl@LTinactivehdl` as the longtable version of `\adl@ihdashline` and `\adl@inactivehdl`. `\adl@LTthdlrow` These two macros, the main part of `\hdashline`, are redefined to make it possible that `\hdashline` can be broken into two part by TeX's page breaker.

<sup>20</sup>In the original `longtable`, a sequence of three `\hline`-s are not recognized. This buggy feature is fixed in this implementation.

These three macros call a common routine `\adl@LThdline` after defining `\adl@LThdlrow` which makes a row of horizontal (dash) line drawn by `\multispan` and `\leaders\hrule` or `\adl@hcline` [*dash*]/*gap*].

Note that we define `\adl@LThdashline` to make `\adl@hdashline` \let-equal to it in `longtable` environments because its version without `longtable` performs a part of the job done by `\adl@LThdline` as shown soon.

```

788
789 %% Compatibility with longtable: horizontal lines and p-boxes
790
791 \def\LT@hline{\noalign{\ifnum0='}\fi
792     \gdef\adl@LThdlrow{\multispan{\LT@cols}\unskip
793         \leaders\hrule\@height\arrayrulewidth\hfill\cr}%
794     \adl@LThdline}
795 \def\adl@LThdashline#1{\noalign{\ifnum0='}\fi
796     \@ifnextchar[%
797         {#1}%
798         {#1[\dashlinedash/\dashlinegap]}}
799 \def\adl@LTihdashline[#1/#2]{%
800     \gdef\adl@LThdlrow{\multispan{\LT@cols}\unskip
801         \adl@hcline\z@[#1/#2]}%
802     \adl@LThdline}
803 \def\adl@LTinactivehdl[#1/#2]{%
804     \gdef\adl@LThdlrow{\multispan{\LT@cols}\unskip
805         \leaders\hrule\@height\arrayrulewidth\hfill\cr}%
806     \adl@LThdline}
807

```

`\adl@LThdline` The macro `\adl@LThdline` called by above three macros first inserts a vertical penalty 10000 to inhibit page break between the horizontal line and preceding row. Then it inserts `\adl@LTxhline` `\vskip-\arrayrulewidth` with another break inhibitor if `\ADLnullwidehline` is in effect, or adds the pseudo row information *connect*(`\arrayrulewidth`) to  $R^{L/R}$  by `\adl@hline`<sup>21</sup>. Next, it draw a horizontal (dash) line by `\adl@LThdlrow` and checks if the following control sequence is `\hline` or `\hdashline` by `\futurelet` and `\adl@LTxhline`. If `\hline` or `\hdashline` is the next token, `\adl@LTixhline` is called to insert a vertical penalty of `-\@medpenalty` and a vertical space of `\doublerulesep`. The macro `\adl@LTixhline` also adds *disconnect*(`\doublerulesep`) to  $R^{L/R}$  and makes `\adl@LThdlrow` void. Otherwise, `\adl@LThdline` inserts a vertical penalty of `-\@lowpanalty` and a vertical space of `-\arrayrulewidth` and draws the horizontal (dash) line again by `\adl@LThdlrow`. Thus a page can be broken between two overlaid horizontal (dash) lines<sup>22</sup>. Two pseudo row information, *discard*(`-\arrayrulewidth`) for the negative vertical space which may be discarded and *connect*(`\arrayrulewidth`) for the second horizontal line, are also added to  $R^{L/R}$ .

```

808 \def\adl@LThdline{\penalty\@M

```

<sup>21</sup>Or do noting if inactive and thus it is \let-equal to `\@gobbletwo`.

<sup>22</sup>If the page is broken, the horizontal line at the beginning of the succeeding page has a width even if `\ADLnullwidehline` is in effect.

```

809      \ifadl@zwhrule \vskip-\arrayrulewidth \penalty\@M
810      \else          \adl@hline\adl@connect\arrayrulewidth \fi
811      \ifnum0='{ \fi}%
812      \adl@LThdlrow
813      \noalign{\ifnum0='}\fi
814      \futurelet\@tempa\adl@LTxhline}
815 \def\adl@LTxhline{\ifx\@tempa\hline \adl@LTixhline
816      \else\ifx\@tempa\hdashline \adl@LTixhline
817      \else \penalty-\@lowpenalty \vskip-\arrayrulewidth
818              \adl@hline\adl@discard{-\arrayrulewidth}%
819              \adl@hline\adl@connect\arrayrulewidth
820      \fi\fi \ifnum0='{ \fi}%
821      \adl@LThdlrow \noalign{\penalty\@M}}
822 \def\adl@LTixhline{\penalty-\@medpenalty \vskip\doublerulesep
823      \adl@hline\relax\doublerulesep \global\let\adl@LThdlrow\@empty}
824

```

`\adl@LTstartpbox` Macros for opening/closing p-boxes are fairly simple. The macro `\adl@LTstartpbox{<w>}` is `\let`-assigned to `\@@startpbox` by `\LT@make@row` to open a p-box of  $w$  wide by our own `\adl@act@@startpbox` and performs a footnote related operation introduced by `longtable`, when `array` is not in use. Note that if `array` is in use, a p-box is opened by codes embedded in `\@preamble` and its initialization is done by `\@startpbox = \LT@startpbox`, unnecessitating our own version of opening macros.

On the other hand, the closing macro `\adl@LTendpbox` for p(or d)-boxes is `\let`-equal to `\@endpbox` and `\@@endpbox` for the cases with/without `array`, and performs the footnote operations after doing our own ones by `\adl@act@@endpbox`. Similarly, `\adl@LTendmbox` for m-boxes is `\let`-equal to `\adl@endmbox` and performs our own operations by `\adl@@endmbox` in which the original definition of `\adl@enmbox` is kept.

```

825 \def\adl@LTstartpbox#1{%
826      \adl@act@@startpbox{#1}\let\@footnotetext\LT@p@ftntext}
827 \def\adl@LTendpbox{\adl@act@@endpbox \the\LT@p@ftn \global\LT@p@ftn{}}
828 \def\adl@LTendmbox{\adl@@endmbox \the\LT@p@ftn \global\LT@p@ftn{}}
829
830 %%^L

```

#### 4.15.4 First Chunk

`\LT@start` The macro `\LT@start` which puts (first) head and controls the page break of the first page is modified for the following.

- After it inserts a vertical skip `\LTpre`, `\endgraf` is performed so that the skip contributes to `\pagetotal`<sup>23</sup>.
- When the `\box2` is `\vsplit` to get first item of the first chunk, `\vbadness` is saved into `\@tempcnta`, set to 10000 to avoid unnecessary `underfull` message<sup>24</sup>, and restored from `\@tempcnta`.

<sup>23</sup>This modification is necessary for the original `longtable`, or it underestimates the room of the first page and leaves head and foot only.

<sup>24</sup>This is also necessary for the original version.

- The `\dimen` register `\adl@LTpagetotal` is set to `\pagetotal` to know the total height of the items preceding `longtable`. Since the assignment is performed after the inserted `\endgraf` and the intentional page break, it should have real total height.
- The box `\LT@firsthead` is put by `\copy` rather than `\box` because it is referred to in the `\output` routine.

This macro does not have inactive counterpart because the modification shown above is desirable (first two) or not-harmful<sup>25</sup> (last two) to the original version.

```

831
832 %% Compatibility with longtable: first chunk
833
834 \def\LT@start{%
835     \let\LT@start\endgraf
836     \endgraf \penalty\z@ \vskip\LTpre \endgraf
837     \dimen@ \pagetotal
838     \advance\dimen@ \ht\ifvoid\LT@firsthead\LT@head\else\LT@firsthead\fi
839     \advance\dimen@ \dp\ifvoid\LT@firsthead\LT@head\else\LT@firsthead\fi
840     \advance\dimen@ \ht\LT@foot
841     \dimen@ii \vfuzz \@tempcnta \vbadness
842     \vfuzz \maxdimen \vbadness \@M
843     \setbox\tw@ \copy \z@
844     \setbox\tw@ \vsplit \tw@ to \ht \@arstrutbox
845     \setbox\tw@ \vbox {\unvbox \tw@}%
846     \vfuzz \dimen@ii \vbadness \@tempcnta
847     \advance\dimen@ \ht
848         \ifdim \ht \@arstrutbox > \ht \tw@ \@arstrutbox \else \tw@ \fi
849     \advance\dimen@ \dp
850         \ifdim \dp \@arstrutbox > \dp \tw@ \@arstrutbox \else \tw@ \fi
851     \advance\dimen@ -\pagegoal
852     \ifdim \dimen@ > \z@ \vfil \break \fi
853     \global \adl@LTpagetotal \pagetotal
854     \global \@colroom \@colht
855     \ifvoid \LT@foot \else
856         \advance \vsize - \ht \LT@foot
857         \global \advance \@colroom - \ht \LT@foot
858         \dimen@ \pagegoal \advance \dimen@ - \ht \LT@foot \pagegoal \dimen@
859         \maxdepth \z@
860     \fi
861     \copy \ifvoid \LT@firsthead \LT@head \else \LT@firsthead \fi
862     \output {\LT@output}}
863
864 %%^L

```

#### 4.15.5 Output Routine

`\adl@org@LT@output`    The output routine is the heart of the `longtable` compatible implementation. The macro  
`\LT@output`            `\LT@output` which is set to `\output` by `\LT@start` is modified from its original (and thus

---

<sup>25</sup>Logically, at least.

inactive) version `\adl@org@LT@output` as follows.

- Three fractions of the original version to compile the final output image of the table portion into `\box255` or the main vertical list are modified to set the image into `\box255` unconditionally and to call `\adl@LTdraw<foot><tail>` which is the real heart of the compatible implementation. The argument `<foot>` is `\LT@foot` or `\LT@lastfoot` according to the portion of the `longtable` to be output. The argument `<tail>` is `\vss` if the last item is it which is not included in `\box255` yet, or `\@empty` otherwise. Since `\adl@LTdraw` builds final output image drawing vertical (dash) lines in `\box255`, it is put to the main vertical list if the `longtable` portion is the last one.
- Since the boxes `\LT@head`, `\LT@foot` and `\LT@lastfoot` are referred to in `\adl@LTdraw`, they are put by `\copy` rather than `\box`.

```

865
866 %% Compatibility with longtable: output routine
867
868 \let\adl@org@LT@output\LT@output
869 \def\LT@output{%
870     \ifnum\outputpenalty <-\@Mi
871         \ifnum\outputpenalty > -\LT@end@pen
872             \LT@err{floats and marginpars not allowed in a longtable}\@ehc
873         \else
874             \setbox\z@\vbox{\unvbox\@cclv}%
875             \ifdim \ht\LT@lastfoot>\ht\LT@foot
876                 \dimen@ \pagegoal
877                 \advance\dimen@-\ht\LT@lastfoot
878                 \ifdim\dimen@<\ht\z@
879                     \setbox\@cclv\vbox{\unvbox\z@\copy\LT@foot}%
880                     \adl@LTdraw\LT@foot\vss
881                     \@makecol
882                     \@outputpage
883                     \setbox\z@\vbox{\copy\LT@head}%
884                 \fi
885             \fi
886             \global\@colroom\@colht
887             \global\vsizel\@colht
888             \setbox\@cclv\vbox{\unvbox\z@
889                 \copy\ifvoid\LT@lastfoot\LT@foot\else\LT@lastfoot\fi}%
890             \adl@LTdraw\LT@lastfoot\@empty \box\@cclv
891         \fi
892     \else
893         \setbox\@cclv\vbox{\unvbox\@cclv\copy\LT@foot}%
894         \adl@LTdraw\LT@foot\vss
895         \@makecol
896         \@outputpage
897         \global\vsizel\@colroom
898         \copy\LT@head
899     \fi}
900

```



`\adl@LTdraw` The macro `\adl@LTdraw<foot><tail>` draws vertical (dash) lines onto the image in `\box255`.  
`\adl@LTinit` First it measures the total height  $H$  (`\adl@totalheight`) of `longtable` rows in `\box255`  
`\adl@LTheadL` and the total height  $H_b$  (`\@tempdima`) of its *body* which consists of the rows without the  
`\adl@LTheadR` header and footer, as follows where  $H_{255}$ ,  $H_h$  and  $H_t$  are the height plus depth of `\box255`  
`\adl@LTfootL` and the effective header and footer of the page respectively.  
`\adl@LTfootR`

$$\begin{aligned}
T &= \begin{cases} \text{\adl@LTpagetotal} & \text{if \ifadl@LTfirstpage} \\ 0 & \text{otherwise} \end{cases} \\
t &= \begin{cases} \text{\topskip glue} & \text{if longtable is the first item of the page} \\ & (\neg(\text{\ifadl@firstpage} \wedge T > 0)) \\ 0 & \text{otherwise} \end{cases} \\
H &= H_{255} - t - T \\
H_b &= H - H_h - H_t
\end{aligned}$$

The hard part is to measure  $t$  because it is not `\topskip` but that minus the first box of `\box255`. Thus we do not measure  $t$  but remove it from the box by the following tricky way. First we copy `\box255` items into `\box0` adding a `\hrule` of 1sp high as its first item. Then `\box0` is `\vsplit` to 1sp setting `\splittopskip` to 0. Since the `\topskip` glue is the first item of `\box255` and the `\vsplit` discards it at the breakpoint, `\box0` must have all the items in `\box255` lead by 0 (`\splittopskip`) glue rather than `\topskip` glue. Thus the height of `\box0` is  $H_{255} - t$ .

Subtraction of  $H_h$  and  $H_t$  is done by the macro `\adl@LTinit{<hf>}<box>`, where `<hf>` is `head` or `foot` and `<box>` is one of `\LT@firsthead`, `\LT@thead` and `<foot>` (`\LT@lastfoot` or `\LT@foot`). This macro also copies the contents of weirdly named structure such as `\adl@rowsL\LT@thead` into `\adl@LTheadL` and so on<sup>26</sup> if `<box>` is not void. Otherwise, `\adl@LTheadL` etc. is kept to their initial value, `\@empty`.

Next, we make rows for vertical lines by `\adl@makev1rL/R` after extracting the leading part of  $R^{L/R}$  corresponding to the *body* by the macro `\adl@LTsplit<RL/R><RhL/R><RfL/R>`, where  $R_h^{L/R}$  and  $R_f^{L/R}$  are `\adl@LTheadL` and so on. Since the macro defines `\adl@rows` given to `\adl@makev1L/R` to the sequence of  $R_h^{L/R}$ , the extracted part of  $R^{L/R}$  and  $R_f^{L/R}$ , the rows for vertical lines for all the rows including header and footer are build in `\adl@v1rowL` and `\adl@v1rowR` as in the ordinary case without `longtable`.

Then the rows are put into `\box0` by calling `\LT@bchunk` with `\adl@drawv1` (line drawing) and `\LT@save@row` (column widths adjustment), saving/restoring counters `\LT@rows` and `\c@LT@chunks` which `\LT@bchunk` globally updates. Since we refer to potentially immature `\LT@save@row` here, some weird looking vertical lines could be drawn but the result after convergence should be correct. Finally, the contents of `\box255` followed by the vertical lines in `\box0` are put back into `\box255` keeping its original depth and adding `<tail>` (`\vss` or nothing) to its end.

```

901 \def\adl@LTdraw#1#2{%
902     \@tempswatrue
903     \ifadl@LTfirstpage\ifdim\adl@LTpagetotal>\z@\@tempswafalse \fi\fi
904     \if@tempswa
905         \setbox\z@\vbox{\hrule height1sp\unvcopy\@cclv}

```

<sup>26</sup>Copying by `\edef` can be replaced by `\let` with many `\expandafter` but it is not comprehensible.

```

906          \splittopskip\z@
907          \setbox\@ne\vsplit\z@ to1sp\relax
908          \@tempdima\ht\z@
909      \else   \@tempdima\ht\@cclv \fi
910      \advance\@tempdima\dp\@cclv
911      \adl@totalheight\@tempdima
912      \let\adl@LTheadL\@empty \let\adl@LTheadR\@empty
913      \let\adl@LTfootL\@empty \let\adl@LTfootR\@empty
914      \ifadl@LTfirstpage
915          \global\adl@LTfirstpagefalse
916          \advance\@tempdima-\adl@LTpagetotal
917          \adl@totalheight\@tempdima
918          \ifvoid\LT@firstthead
919              \adl@LTinit{head}\LT@head
920          \else   \adl@LTinit{head}\LT@firstthead
921          \fi
922      \else   \adl@LTinit{head}\LT@head \fi
923      \ifvoid#1%
924          \adl@LTinit{foot}\LT@foot
925      \else   \adl@LTinit{foot}\#1\fi
926      \let\adl@vl\relax \def\adl@discard{\adl@connect}%
927      \def\adl@vlrow{\adl@currentcolumn\@ne
928          \adl@LTsplit\adl@rowsL\adl@LTheadL\adl@LTfootL
929          \let\adl@addvl\adl@addvL
930          \adl@makevlrL \let\adl@vlrowL\adl@vlrow
931      \def\adl@vlrow{\adl@currentcolumn\adl@columns
932          \adl@LTsplit\adl@rowsR\adl@LTheadR\adl@LTfootR
933          \let\adl@addvl\adl@addvR
934          \adl@makevlrR \let\adl@vlrowR\adl@vlrow
935      \let\adl@vl\adl@@vl
936      \@tempcnta\LT@rows
937      \LT@bchunk \adl@drawvl
938      \LT@save@row\cr \egroup \setbox\@ne\lastbox \unskip \egroup
939      \global\advance\c@LT@chunks\m@ne
940      \global\LT@rows\@tempcnta
941      \@tempdima\dp\@cclv
942      \setbox\@cclv\vbox{\unvbox\@cclv \box\z@ \vskip-\@tempdima
943          \hrule\@width\z@\@height\z@\@depth\@tempdima#2}}
944      \def\adl@LTinit#1#2{\ifvoid#2\else
945          \advance\@tempdima-\csname\string\adl@LTth\string#2\endcsname sp%
946          \expandafter\edef\csname adl@LT#1L\endcsname{%
947              \csname\string\adl@rowsL\string#2\endcsname}%
948          \expandafter\edef\csname adl@LT#1R\endcsname{%
949              \csname\string\adl@rowsR\string#2\endcsname}\fi}
950

```

$\backslash\text{adl@LTsplit}$  The macro  $\backslash\text{adl@LTsplit}\langle R^{L/R}\rangle\langle R_h^{L/R}\rangle\langle R_f^{L/R}\rangle$  moves leading elements in  $R^{L/R}$  into  $R'$   
 $\backslash\text{adl@LTxsplit}$  ( $\backslash\text{adl@rows}$ ) until total heights of the elements summed in  $h$  ( $\backslash\text{@tempdimb}$ ) reaches to  $H_b$   
 $\backslash\text{adl@LTrowrelax}$   
 $\backslash\text{adl@LTrowdiscard}$   
 $\backslash\text{adl@LTysplit}$   
 $\backslash\text{adl@LTisplit}$   
 $\backslash\text{adl@LTiisplit}$   
 $\backslash\text{adl@LTsplitend}$

(\@tempdima)<sup>27</sup> by a straightforward loop with the macros \adl@LTisplit to fetch the  $i$ -th element and \adl@LTisplit to get  $h_i$ . Before moving, however, we have to remove discardable item(s)<sup>28</sup> from the top of  $R^{L/R}$ . Since an element for a discardable item is *disconnect* (\relax) or *discard* (\adl@discard), we check the first part of the element by \ifx-comparison with \adl@LTrowrelax and \adl@LTrowdiscard whose bodies are \relax and \adl@discard if the longtable portion does not have a header ( $R_h^{L/R}$  is \@empty). Otherwise, the discardable item was not discarded because the first item of the page is not it but the header.

Note that since moving from  $R^{L/R}$  to  $R'$  is done by \edef and \adl@discard is \def-ined as \adl@connect in \adl@LTdraw, non-discarded *discard* transforms into *connect* in  $R'$ . Also note that since the remaining part of  $R^{L/R}$  is \def-ined as the body of \@tempb which is globally \let-assigned to  $R^{L/R}$  again, \adl@discard survives in the new  $R^{L/R}$ .

```

951 \def\adl@LTsplit#1#2#3{\def\adl@rows{}\@tempdima\z@
952     \expandafter\adl@LTxsplit#1\@nil;%
953     \edef\adl@rows{#2\adl@rows#3}%
954     \global\let#1\@tempb}
955 \def\adl@LTxsplit#1;{\def\@tempa{#1}%
956     \ifx\@tempa\@nnil \def\@tempb{}\let\next\relax
957     \else\ifx\adl@LtheadL\@empty \def\next{\adl@LTysplit#1}%
958     \else \def\next{\adl@LTisplit#1;}\fi \fi
959     \next}
960 \def\adl@LTrowrelax{\relax}
961 \def\adl@LTrowdiscard{\adl@discard}
962 \def\adl@LTysplit(#1/#2){\def\@tempa{#1}%
963     \ifx\@tempa\adl@LTrowrelax \let\next\adl@LTxsplit
964     \else\ifx\@tempa\adl@LTrowdiscard \let\next\adl@LTxsplit
965     \else \def\next{\adl@LTisplit(#1/#2);}\fi \fi
966     \next}
967 \def\adl@LTisplit#1;{\def\@tempa{#1}%
968     \ifx\@tempa\@nnil \def\@tempb{}\let\next\relax
969     \else\ifdim\@tempdima<\@tempdima
970         \adl@LTisplit#1\let\next\adl@LTisplit
971     \else \def\next{\adl@LTsplitend#1;}\fi \fi
972     \next}
973 \def\adl@LTisplit(#1/#2){\edef\adl@rows{\adl@rows(#1/#2);}%
974     \advance\@tempdima#2sp}
975 \def\adl@LTsplitend#1;\@nil;{\def\@tempb{#1;}}
976 \fi
977
978 %^L

```

<sup>27</sup>Although  $h$  must become  $H_b$  exactly in usual case, we stop the loop when  $h \geq H_b$  to avoid accidental overrun in unusual cases.

<sup>28</sup>Must be only one but the implementation allows two or more.

## 4.16 Compatibility with colortbl

The implementation to make `arydshln` compatible with `colortbl` consists of the following three (almost independent) issues.

**Cell coloring** is the easiest part because it does not affect dash line drawing. Another reason of the easiness is that `colortbl` packs each cell in a box to measure its height for painting in the modified version of `\@classz`. Thus we do not need to code `\@classz` for both of `colortbl` and `arydshln`, but may sneak our own height/depth measurement into `\@classz` of `colortbl`. Almost everything we have to pay attention to is the compatibility of the initialization and finalization of `colortbl` and `arydshln`.

**Horizontal line coloring** is relatively easy because it is almost enough to insert coloring macro `\CT@arc@` before the line drawing. A little bit complicated part is the gap coloring which is done by drawing a solid line of gap color before dash line is drawn.

**Vertical line coloring** is the hardest part but almost everything is done in previous sections to attach dash/gap color to each vertical line segment  $e_j^i$  in the list  $C_i^L$  and  $C_i^R$  of the  $i$ -th row information  $r_i$ . What we do here is to fix the bugs of `\arrayrulecolor` and `\doublerulesepcolor` in `colortbl` implementation and to add `\dashgapcolor`. If you put `\arrayrulecolor` in `>\{...\}` construct to specify the color of the vertical lines following the construct as the manual of `colortbl` says, you will have an error message “Misplaced `\noalign`” because the macro is expanded with `\noalign` in a column body. Even if you somehow remove `\noalign` to avoid the error, you will have a mysterious line coloring as follows:

- If you have `\arrayrulecolor` before the `\array/\tabular` starts, `\arrayrulecolor` in the preamble has no effect to vertical lines but decides the color of horizontal lines except for those at the top of the environment. Additional `\arrayrulecolor` at the beginning of a row has no effect to vertical lines (as expected) but decides horizontal lines following it (also as expected). The effect of `\doublerulesepcolor` is same as `\arrayrulecolor`.
- Otherwise, i.e. without `\arrayrulecolor` outside the environment, `\arrayrulecolor` in the preamble decides the color of vertical and horizontal lines except for verticals preceding columns in the first row and horizontals at the top of the environment. Additional `\arrayrulecolor` at the beginning of a row decides all the vertical and horizontal lines following it. On the other hand, `\doublerulesepcolor` acts as if `\doublerulesepcolor{white}` is done outside the environment.

The reason of the mysterious behavior is as follows. An `\arrayrulecolor`, which globally `\def`-ines a macro `\CT@arc@` with a body containing `\color`, in the preamble is not expanded nor evaluated in the preamble construction phase but done when the first (and succeeding) row is build. On the other hand, `\CT@arc@` attached to vertical line drawing is expanded in the preamble construction phase. Thus if `\CT@arc@` has been defined before the environment starts, vertical lines are colored following the outside definition. Otherwise, since `\CT@arc@` is `\let`-equal to `\relax`, it remains unchanged in the preamble construction phase and expanded when each row is build

referring to its definition that `\arrayrulecolor` modifies in the row building phase. Since the macro `\CT@drsc@` defined by `\doublerulesepcolor` is examined if it is `\relax` or not in the preamble construction phase, `\doublerulesepcolor` in the preamble has no effect regardless the existence of the outside definition.

Thus we have to expand and evaluate `\arrayrulecolor` and `\doublerulecolor` in the preamble construction phase to define `\CT@arc@` and `\CT@drsc@`. We also have to initialize `\CT@arc@` as an expandable but non-operative token (e.g. a macro with a body of `\relax` as we do) to make it is expanded in the preamble construction phase rather than the row building.

#### 4.16.1 Initialization, Cell Coloring and Finalization

`\CT@arc@` First of all, we initialize the macro `\CT@arc@`, which will be `\def`-ined as `\color` to specify the color of solid lines and dash segments by `\arrayrulecolor`, with a body of `\relax` because it will be referred to by the vertical line drawing process even if `colortbl` is not in use. We also initialize the macro `\adl@dashgapcolor` for the color of gaps of dash lines similarly. Note that these macros are not `\let`-equal to `\relax` but have bodies of `\relax` so that they are replaced with `\relax` in the preamble construction phase rather than surviving with their own name.

```
979
980 %% Compatibility with colortbl
981
982 \def\CT@arc@{\relax}
983 \def\adl@dashgapcolor{\relax}
```

Next we examine if `colortbl` is in use by `\ifpackageloaded`, and skip everything if not, or we have some errors especially when `array` is not in use.

```
984 \@ifpackageloaded{colortbl}\@tempswatrue\@tempswafalse
985 \if@tempswa
```

`\adl@org@inactivate` Then we redefine `\adl@inactivate` and `\adl@activate` referring their original version  
`\adl@org@activate` `\adl@org@inactivate` and `\adl@org@activate` so that they make `\CT@setup` `\let`-equal  
`\adl@inactivate` to its original version `\adl@CT@setup` if `\ADLinactivate` is in effect, or to our own ver-  
`\adl@activate` sion `\adl@act@CT@setup` which will be defined soon. New `\adl@activate` also *inactivates*  
`\CT@setup` `\@endpbox` because our own one for column height/depth measuremnt is inappropriate with  
`\@endpbox` `colortbl` as explained soon.

```
986 \let\adl@org@inactivate\adl@inactivate
987 \let\adl@org@activate\adl@activate
988 \def\adl@inactivate{\adl@org@inactivate \let\CT@setup\adl@CT@setup}
989 \def\adl@activate{\adl@org@activate \let\CT@setup\adl@act@CT@setup
990 \let\@endpbox\adl@org@endpbox}
991
```

`\adl@CT@setup` Cell coloring is done by `\@classz` preamble of `colortbl` in which a column is packed in  
`\CT@setup` `\box0`. On the other hand, our own `\@classz` one with `array` packs the column in `\adl@`  
`\adl@act@CT@setup` box so that we measure its height and depth. Thus we have choices; to insert height/depth

measurement into `colortbl`'s version; or to insert coloring into our own version. Since the code of height/depth measurement is much simpler than the coloring, we choose the first way. Thus the macro `\adl@act@CT@setup`, which is `\let`-equal to `\CT@setup` and is invoked from `\@classz` preamble after the column is packed into `\box0`, measures the height and depth of `\box0` and sets `\adl@height` and/or `\adl@depth` to them if they break the records as `\adl@colhtdp` does with `\adl@box`, after it invokes its original version `\adl@CT@setup`. Note that we compare `\adl@height` with the height of `\box0` plus `\minrowclearance` because it is the real height. Also note that we could insert the measurement code into the modified version of `colortbl`'s `\@classz` placing it just before the `\box0` is put where `\ht0` plus `\minrowclearance` is calculated, but did not because the author wished to make it clear that `\@classz` is modified only for the bug fix of `\arrayrulecolor` and `\doublerulesepcolor` (and to introduce `\dashgapcolor`).

```

992 \let\adl@CT@setup\CT@setup
993 \def\CT@setup{\adl@CT@setup
994     \@tempdima\ht\z@ \advance\@tempdima\minrowclearance
995     \ifdim\adl@height<\@tempdima \global\adl@height\@tempdima \fi
996     \ifdim\adl@depth<\dp\z@ \global\adl@depth\dp\z@\fi}
997 \let\adl@act@CT@setup\CT@setup
998

```

`\adl@activatepbox` Another job for cell coloring is to make `\CT@x@color` ( $x \in \{\text{cell}, \text{column}, \text{do}\}$ ) `\let`-equal to `\relax` before the body of `\multicolumn` is put so that the `\columncolor` in the environment preamble does not affect the `\span`-ned column. Note that resetting `\CT@cell@color` will be unnecessary (but safe) because it is always reset after its invocation. Also note that resetting `\CT@row@color` in `colortbl`'s `\multicolumn` is a buggy feature because it should be effective, and thus we remove it. Although we have our own `\multicolumn` for dash lines, we keep it unchanged. Instead we redefine `\adl@activatepbox`, which is usually `\relax` with `array`, to do the color resetting to minimize recoding.

```

999 \def\adl@activatepbox{\let\CT@cell@color\relax
1000     \let\CT@column@color\relax
1001     \let\CT@do@color\relax}
1002

```

`\adl@CT@start` Yet another job is the save/restore of color information at the beginning and end of the environment. Since this is done by `\CT@start` and `\CT@end`, we modify them to save/restore `\adl@dashgapcolor` to/from `\adl@dashgapcolor@save` referring their original version `\adl@CT@start` and `\adl@CT@end`. We also modify our own `\endarray` and its shorthand active version `\endArray` so that `\CT@end` is invoked at the end of environment together with `\@arrayright` if it is defined. Note that we may not modify `\endtabular` because it refers `\endarray`. Also note that `\CT@start` is invoked from `\@tabarray` which we keep unchanged.

```

1003 \let\adl@CT@start\CT@start
1004 \def\CT@start{\adl@CT@start \let\adl@dashgapcolor@save\adl@dashgapcolor}
1005 \let\adl@CT@end\CT@end
1006 \def\CT@end{\adl@CT@end \global\let\adl@dashgapcolor\adl@dashgapcolor@save}

```

```

1007 \def\endarray{\adl@endarray \egroup \adl@arrayrestore \CT@end \egroup
1008         \csname @arrayright\endcsname}
1009 \ifx\adl@notdefinable\undefined \let\endArray\endarray \fi
1010

```

#### 4.16.2 Horizontal Line Coloring

`\hline` To color `\hline` and inactivated `\hdashline`, we modify our own `\hline` and `\adl@inactivehdl` inserting the line coloring macro `\CT@arc@` before drawing by `\hrule` and `\adl@ixhline` pushing the coloring/drawing into a group. We also modify `\adl@ixhline` to draw a colored horizontal rule of `\doublerulesep` wide with the color defined in `\CT@drsc@` if it is not `\relax`, rather than to insert a vertical skip. Note that the `\cline` coloring is done by `colortbl`'s `\cline` renamed as `\adl@org@cline` and invoked from our own one.

```

1011 \def\hline{\noalign{\ifnum0='}\fi
1012         \ifadl@zwhrule \vskip-\arrayrulewidth
1013         \else \adl@hline\adl@connect\arrayrulewidth \fi
1014         {\CT@arc@ \hrule\@height\arrayrulewidth}%
1015         \global\adl@finaldepth\z@
1016         \futurelet\@tempa\adl@xhline}
1017 \def\adl@inactivehdl[#1/#2]{\ifadl@zwhrule \vskip-\arrayrulewidth \fi
1018         {\CT@arc@ \hrule\@height\arrayrulewidth}%
1019         \futurelet\@tempa\adl@xhline}
1020 \def\adl@ixhline{\ifx\CT@drsc@\relax \vskip \else
1021         \CT@drsc@\hrule\@height \fi \doublerulesep}%
1022         \adl@hline\relax\doublerulesep}

```

`\adl@ihdashline` To draw a horizontal dash line with colored dashes and also colored gaps, we drastically modified `\adl@ihdashline` for `\hdashline` and `\adl@cdline` for `\cdashline`. First, they invoke `\adl@hcclinesetup` that makes the prefix of a `\multispan`-ned row from the first to last columns for `\hdashline` or given columns for `\cdashline`. Then the line is drawn by the modified version of `\adl@hccline`. We have to declare these macros are active ones again.

```

1023 \def\adl@ihdashline[#1/#2]{\adl@hcclinesetup\@ne\adl@columns
1024         \adl@hccline\z@[#1/#2]%
1025         \noalign{\ifnum0='}\fi
1026         \futurelet\@tempa\adl@xhline}
1027 \let\adl@act@ihdashline\adl@ihdashline
1028 \def\adl@cdline[#1-#2]{\ifadl@zwhrule \vskip-\arrayrulewidth \fi
1029         \adl@hcclinesetup{#1}{#2}%
1030         \adl@hccline{-\arrayrulewidth}}
1031 \let\adl@act@cdline\adl@cdline

```

`\adl@hcclinesetup` The macro `\adl@hcclinesetup<f><t>` makes the prefix of a `\multispan`-ned row from the column  $f$  to  $t$  and `\global`-ly defines it as `\@gtempa`. This is done by a code very similar to `\adl@cdliena` (and thus L<sup>A</sup>T<sub>E</sub>X-2.09's `\cline`) but the invocation of `\adl@hccline` is removed from `\adl@cdliena` and `\adl@cdlineb`, one of which is `\@gtempa`.

```

1032 \def\adl@hclinesetup#1#2{\global\adl@cla#1\relax
1033         \global\advance\adl@cla\m@ne
1034         \ifnum\adl@cla>\z@ \global\let\@gtempa\adl@cdlinea
1035         \else \global\let\@gtempa\adl@cdlineb\fi
1036         \global\adl@clb#2\relax
1037         \global\advance\adl@clb-\adl@cla \ifnum0='{ \fi}}
1038 \def\adl@cdlinea{\multispan\adl@cla &\multispan\adl@clb \unskip}
1039 \def\adl@cdlineb{\multispan\adl@clb \unskip}

```

`\adl@hcline` The modified version of `\adl@hcline<w>[<d>/<g>]` draws a colored horizontal dash line of dash size  $d$  and gap size  $g$  and insert vertical skip of  $w$ . First it `\span-s` columns by `\@gtempa` and checks if the body of `\adl@dashgapcolor` is something other than `\relax`. If so, i.e. it has `\color`, `\adl@paintdashgap` is invoked to draw a horizontal rule of `\color` by `\leaders` as the background of the dash line, to insert `\nobreak` (for `longtable`) and a negative space for canceling the width of the rule, and to `\span` the columns again. Then `\adl@hcline` draws the colored dash line, over the background if the gaps are colored, by inserting `\CT@arc@` before the invocation of `\adl@draw`.

```

1040 \def\adl@hcline#1[#2/#3]{\@gtempa
1041         \ifx\adl@dashgapcolor\adl@nocolor \else \adl@paintdashgap \fi
1042         {\@tempdima#2\relax \@tempdimb#3\relax
1043         \CT@arc@ \adl@draw\adl@vrule\hskip\hbox}\cr
1044         \noalign{\global\adl@finaldepth\z@ \ifdim#1=\z@ \else
1045         \ifadl@zwhrule\else \vskip#1\fi\fi}}
1046 \def\adl@paintdashgap{\adl@dashgapcolor
1047         \leaders\hrule\@height\arrayrulewidth\hfill}\cr
1048         \noalign{\penalty\@M \vskip-\arrayrulewidth}\@gtempa}
1049

```

### 4.16.3 Vertical Line Coloring

`\arrayrulecolor` A bug of colortbl's `\arrayrulecolor` and `\doublerulesepcolor` is that they are defined like;

`\CT@arc@`

`\doublerulesepcolor` `\CT@drsc@` `\ifdim\baselineskip=\z@ \noalign \fi{\gdef\CT@arc@{\color...}}`

`\dashgapcolor` This aims to do `\noalign{\gdef...}` in `array/tabular` and do `{\gdef...}` outside but has two problems: First, if they are in `>{...}` construct, they are expanded with `\noalign` inappropriately when the argument of `>` is expanded. Second, they may appear at a place where `\baselineskip` is 0 but is outside of `array/tabular` and will cause the misplaced `\noalign` error. To solve the second problem, we introduced `\adl@noalign` which is set to `\noalign` in the environment by our own `\@array`, and `\relax` outside. We also introduced `\adl@defcolor<cs><opt>` for the common job to define `<cs>` as `\color` with `<opt>`, in `\noalign` if necessary, by `\adl@idefcolor`. Thus `\arrayrulecolor` and `\doublerulesepcolor` are modified to define `\CT@arc@` and `\CT@drsc@` using `\adl@defcolor`, and our own `\dashgapcolor` is defined similarly to define `\adl@dashgapcolor`. Another macro `\nodashgapcolor` to nullify `\dashgapcolor` is also defined with `\adl@noalign` to reset `\adl@dashgapcolor` to `\relax`.



```

1050 \def\arrayrulecolor{\adl@defcolor\CT@arc@}
1051 \def\doublerulesepcolor{\adl@defcolor\CT@drsc@}
1052 \def\dashgapcolor{\adl@defcolor\adl@dashgapcolor}
1053 \def\adl@defcolor#1#2#{\adl@idefcolor{#1}{#2}}
1054 \def\adl@idefcolor#1#2#3{\adl@noalign{\gdef#1{\color#2{#3}}}}
1055 \let\adl@noalign\relax
1056 \def\nodashgapcolor{\adl@noalign{\gdef\adl@dashgapcolor{\relax}}}
1057

```

**\@classz** The tougher bug of colortbl is the expansion timing of `\arrayrulecolor` and `\doublerulesepcolor` in a `>`-argument. We have to modify `\@classz` to extract them from `\toks`  
**\adl@act@classz** `\@tempcnta` as its original version does for `\columncolor`. Thus we inserted the invocation of `\adl@extract@arc` for `\arrayrulecolor`, `\adl@extract@drsc` for `\doublerulesepcolor`, and `\adl@extract@dgc` for `\dashgapcolor` just after the invocation of `\CT@extract`. Note that the other part of `\@classz` is not modified logically, but done for author's preference of indentation. Also note that both `\adl@act@classz` and `\adl@org@classz` are `\let`-equal to the modified `\@classz` because we have to be bug free even if `\ADLinactive` is in effect.

```

1058 \def\@classz{\@classx
1059     \@tempcnta\count@ \prepnext@tok
1060     \expandafter\CT@extract\the\toks\@tempcnta\columncolor!\@nil
1061     \expandafter\adl@extract@arc\the\toks\@tempcnta\arrayrulecolor!\@nil
1062     \expandafter\adl@extract@drsc
1063         \the\toks\@tempcnta\doublerulesepcolor!\@nil
1064     \expandafter\adl@extract@dgc\the\toks\@tempcnta\dashgapcolor!\@nil
1065     \@addtopreamble{%
1066         \setbox\z@\hbox\bgroup\bgroup
1067         \ifcase \@chnum
1068             \hskip\stretch{.5}\kern\z@
1069             \d@llarbegin
1070             \insert@column
1071             \d@llarend\hskip\stretch{.5}%
1072         \or \d@llarbegin \insert@column \d@llarend \hfill
1073         \or \hfill \kern\z@ \d@llarbegin \insert@column \d@llarend
1074         \or $\vcenter
1075             \@startpbox{\@nextchar}\insert@column \@endpbox $\%
1076         \or \vtop \@startpbox{\@nextchar}\insert@column \@endpbox
1077         \or \vbox \@startpbox{\@nextchar}\insert@column \@endpbox
1078         \fi
1079         \egroup\egroup
1080         \begingroup
1081             \CT@setup
1082             \CT@column@color
1083             \CT@row@color
1084             \CT@cell@color
1085             \CT@do@color
1086         \endgroup
1087         \@tempdima\ht\z@
1088         \advance\@tempdima\minrowclearance

```

```

1089             \vrule\@height\@tempdima\@width\z@
1090             \unhbox\z@}%
1091         \prepnext@tok}
1092 \let\adl@act@classz\@classz
1093 \let\adl@org@classz\@classz
1094

```

\adl@def@extract The definitions of \adl@extract@ $x$  ( $x \in \{\text{arc}, \text{drsc}, \text{dgc}\}$ ) are quite similar to each other.  
\adl@extract@arc For example \adl@extract@arc is defined as follows.

```

\adl@extract@arc@b \def\adl@extract@arc#1\arrayrulecolor#2#3\@nil{%
    \CT@arc@ \if!#2\toks\@tempcnta{#1}\let\@tempa\relax%
\adl@extract@drsc \else\if[#2%
\adl@extract@drsc@b \def\@tempa{\adl@extract@arc@b{#1}#3\@nil}%
    \CT@drsc@ \else \def\CT@arc@{\color{#2}}%
\adl@extract@dgc \def\@tempa{\adl@extract@arc#1#3\@nil}%
\adl@extract@dgc@b \fi\fi \@tempa}
\adl@dashgapcolor \def\adl@extract@arc@b#1#2]#3{%
    \def\CT@arc@{\color{#2}{#3}}%
    \adl@extract@arc#1}

```

This code extracts *all the* occurrences of `\arrayrulecolor[ $\langle m \rangle$ ]{ $\langle c \rangle$ }` from the token register and \def-ines `\CT@arc@` as `\color[ $\langle m \rangle$ ]{ $\langle c \rangle$ }`. Note that `\CT@extract` does a similar job for `\columncolor` but it mistakingly ignores the possibility that the token register has two or more `\columncolor`<sup>29</sup>. Anyway, if we copy the code above and replace ‘@arc’ with ‘@drsc’, `\arrayrulecolor` with `\doublerulesepcolor`, and `\CT@arc@` with `\CT@drsc@`, we will have `\adl@extract@drsc(@b)` for `\doublerulesepcolor`. The code for `\adl@extract@dgc(@b)` will be also obtained similarly. However, having three relatives for a almost common job is too awful. Thus we introduce;

```
\adl@def@extract<key><umac><cmac>
```

to define the macros `\adl@extract@key` and `\adl@extract@key@b` for the user interface macro `<umac>` in which a color macro `<cmac>` is defined with `\color`. For example, we will obtain `\adl@extract@arc(@b)` shown above by;

```
\adl@def@extract{arc}\arrayrulecolor\CT@arc@
```

Note that `\color` is made `\relax` in the preamble construction phase by `colortbl`’s `\@mkpream` and regain its proper meaning after the phase.

```

1095 \def\adl@def@extract#1#2#3{%
1096     \expandafter\def\csname adl@extract@#1\endcsname##1#2##2##3\@nil{%
1097         \if!##2\toks\@tempcnta{##1}\let\@tempa\relax
1098         \else\if[#2%
1099             \def\@tempa{\@nameuse{adl@extract@#1@b}{##1}##3\@nil}%
1100         \else \def#3{\color{##2}}%
1101         \def\@tempa{\@nameuse{adl@extract@#1}##1##3\@nil}%
1102         \fi\fi \@tempa}

```

---

<sup>29</sup>Fixing this bug is not our business.

```

1103      \expandafter\def\csname adl@extract@#1@b\endcsname##1##2]##3{%
1104          \def#3{\color{##2}{##3}}%
1105          \@nameuse{adl@extract@#1}##1}}
1106 \adl@def@extract{arc}\arrayrulecolor\CT@arc@
1107 \adl@def@extract{drsc}\doublerulesepcolor\CT@drsc@
1108 \adl@def@extract{dgc}\dashgapcolor\adl@dashgapcolor
1109

```

#### 4.16.4 Compatibility with longtable

`\LT@hline` Yet another compatibility issue is to cope with both `longtable` and `colortbl`. We redefine `\adl@LTihdashline` `\LT@hline` and `\LT@inactivehdl` in order to put `\CT@arc@` before line drawing and to push them in a group. Modified `\adl@LTidashline` first invokes `\adl@hclinesetup` and `\adl@LTinactivehdl` open `\noalign` because it is closed by `\adl@hclinesetup`. The contents of `\adl@LThdlrow` for `\adl@LTidashline` is simply `\adl@hcline` because it does `\multispan` now. The macro `\adl@LTixhline` is modified to paint the `\doublerulesep` gap by `\leaders\hrule` with color of `\CT@drsc@` if it is not `\relax`.

```

1110 \ifx\longtable\undefined\else
1111 \def\LT@hline{\noalign{\ifnum0='}\fi
1112     \gdef\adl@LThdlrow{\multispan{\LT@cols}\unskip{\CT@arc@
1113         \leaders\hrule\@height\arrayrulewidth\hfill}\cr}%
1114     \adl@LThdline}
1115 \def\adl@LTidashline[#1/#2]{\adl@hclinesetup\@ne\adl@columns
1116     \noalign{\ifnum0='}\fi
1117     \gdef\adl@LThdlrow{\adl@hcline\z@[#1/#2]}%
1118     \adl@LThdline}
1119 \def\adl@LTinactivehdl[#1/#2]{%
1120     \gdef\adl@LThdlrow{\multispan{\LT@cols}\unskip{\CT@arc@
1121         \leaders\hrule\@height\arrayrulewidth\hfill}\cr}%
1122     \adl@LThdline}
1123 \def\adl@LTixhline{%
1124     \ifx\CT@drsc@\relax \gdef\adl@LThdlrow{\noalign{
1125         \penalty-\@medpenalty \vskip\doublerulesep}}
1126     \else \gdef\adl@LThdlrow{\noalign{\penalty\@M}%
1127         \multispan{\LT@cols}\unskip{\CT@drsc@
1128             \leaders\hrule\@height\doublerulesep\hfill}\cr}\fi
1129     \ifnum0='\fi\adl@LThdlrow \noalign{\ifnum0='}\fi
1130     \adl@hline\relax\doublerulesep \global\let\adl@LThdlrow\@empty}
1131 \fi
1132 \fi

```

## Acknowledgments

The author thanks to Monty Hayes who gave the author the opportunity to make this style, and Weimin Zhang and Takahiro Kubota who pointed out bugs in early versions. He also thanks to the following people; Sebastian Rahtz and Graham Williams who kindly invited the style to T<sub>E</sub>X CTAN and online catalogue compiled by Graham; Peter Ehrbar who

showed the style was incompatible with `array` and kindly accepted the offer to be an alpha-user of v1.4 alone; Zsuzsanna Nagy who reported another incompatibility problem with `colortab`; Ralf Heydenreich who reported the bug causing that glues in a column have no effect; Yaxin Liu who reported the incompatibility bug of `array` and `\ADLinactivate`; Craig Leech who reported the incompatibility problem with `longtable`, which was also reported by Uwe Jehmlich, Torge Thielemann and Florian Weig, and had waited for two years and a half (!) for the solution; Klaus Dalinghaus who reported yet another incompatibility with `colortbl`; Morten Høgholm who reported the bug of `m`-type columns of `array` which had not manifested in five (!! ) years since the author released the first `array`-compatible version; Maïeul Rouquette who reported another bug of `m`-type columns of `longtable` with `array` which had peacefully hidden in the package for eleven years and a half (!!!) since the author made the bug fix shown above carelessly, yet another bug related to `longtable`, and most surprisingly a problem on intersections of horizontal and vertical (dash-)lines which has hidden for 23 years (!!!!) since the very first version of the package; and Hironobu Yamashita who pointed out bugs hidden for 19 years (!!!!!) by which `delarray` did not work.

The base implementation of `array` and `tabular` environments, part of which the author gives new definitions referring original ones, are written by Leslie Lamport as a part of `LATEX-2.09` and `LATEX 2ε` (1997/12/01) to which Johannes Braams and other authors also contributed. The author also refers `array` package (v2.4c) written by Frank Mittelbach and David Carlisle; `colortab` package (v0.9) written by Timothy van Zandt; and `longtable` (v4.10) and `colortbl` (v0.1j) packages written by David Carlisle; to make the style compatible with those packages.

## Index

Italicized number refers to the page where the specification and usage of corresponding entry are described, while underlined is for the implementation of the entry. To find a control sequence, remove prefixes \@, \adl@ and \ifadl@ from its name if it has one of them.

Symbols	
:	4
;	5
\adl@LTth\LT@firsthead	51, 57
\adl@LTth\LT@foot	51, 57
\adl@LTth\LT@head	51, 57
\adl@LTth\LT@lastfoot	51, 57
\adl@rowsL\LT@firsthead	51, 57
\adl@rowsL\LT@foot	51, 57
\adl@rowsL\LT@head	51, 57
\adl@rowsL\LT@lastfoot	51, 57
\adl@rowsR\LT@firsthead	51, 57
\adl@rowsR\LT@foot	51, 57
\adl@rowsR\LT@head	51, 57
\adl@rowsR\LT@lastfoot	51, 57
<b>A</b>	
\AC	7
\adl@act@@endpbox	21, 47, 54
\adl@act@@startpbox	21, 47, 54
\adl@act@@vlineL	21, 47
\adl@act@@vlineR	21, 47
\adl@act@argcr	21, 47
\adl@act@arrayclassz	21, 47
\adl@act@cdline	21, 47, 63
\adl@act@classz	21, 47, 65
\adl@act@ccline	21, 47
\adl@act@cr	21, 47
\adl@act@CT@setup	61
\adl@act@endarray	21, 47
\adl@act@endpbox	21, 47
\adl@act@hline	21, 47
\adl@act@ihdashline	21, 47, 63
\adl@act@tabclassz	21, 47
\adl@activate	20, 61
\adl@activatepbox	29, 62
\@addamp	23
\adl@addv1	36, 40, 41, 57
\adl@addv1L	42, 57
\adl@addv1R	42, 57
\ADLactivate	7, 15
\ADLdrawingmode	6, 45
\ADLinactivate	6, 15
\ADLnoshorthand	7, 46
\ADLnullwide	5, 15
\ADLnullwidehline	8, 15
\ADLsomewide	5, 15
\ADLsomewidehline	8, 15
\afterassignment	22
\adl@argarraydashrule	24, 25, 26, 30, 31
\adl@argcr	20, 21, 32, 47
\Array	46
Array (environment)	7
\array	45
array (environment)	4
array (package)	3, 7
\@array	17, 64
\@@array	18
\adl@Array	45
\adl@array	17
\@arrayclassz	20, 21, 25, 47
\adl@arraydashrule	24, 25, 26, 30, 31
\adl@arrayinit	19, 49
\adl@arrayrestore	36, 38, 47
\@arrayright	36
\@arrayrule	24, 25
\adl@arrayrule	24, 25, 26, 30, 31
\arrayrulecolor	7, 18, 64
\arrayrulewidth	5
\adl@arraysave	19, 47
\@arstrutbox	10, 32
<b>B</b>	
\adl@box	15, 21, 24, 26, 29, 44
<b>C</b>	
\c@LT@chunks	57
\CC@	47
\adl@CC@	47
\cdashline	4, 5, 35
\adl@cdlinea	35, 63
\adl@cdline	20, 21, 35, 47, 63
\adl@cdlineb	35, 63
\cellcolor	7

\@chclass	26		
\@cla	17		
\adl@cla	17, 35		
\adl@class@iiiorvii	25, 26		
\adl@class@start	25, 26		
\@classv	26		
\adl@classv	26		
\adl@classvfordash	26		
\@classz	20, 21, 24, 47, 65		
\@clb	17		
\adl@clb	17, 35		
\cleaders	44		
\cline	4, 33, 47, 63		
\@cline	35		
\adl@colhtdp	21, 22, 26, 27, 28, 29		
\adl@@colhtdp	28, 61		
\color	8, 26, 28		
colortab (package)	3, 7		
colortbl (package)	3, 7		
\adl@colsL	19, 32, 38		
\adl@colsLsave	19, 38		
\adl@colsR	19, 32, 38		
\adl@colsRsave	19, 38		
\columncolor	7		
\adl@columns	16, 23, 38, 49		
\adl@connect	19, 32, 33, 40, 53		
\adl@@connect	40		
\ifadl@connected	14, 38, 42		
counters:			
LTchunksize	8		
\adl@cr	20, 21, 32, 47		
\adl@@cr	32, 36, 50		
\CT@arc@	22, 26, 61, 63, 64, 66		
\CT@cell@color	62		
\CT@column@color	62		
\CT@do@color	62		
\CT@drsc@	63, 64, 66		
\CT@end	62		
\adl@CT@end	62		
\CT@row@color	62		
\CT@setup	61		
\adl@CT@setup	61		
\CT@start	62		
\adl@CT@start	62		
\current@color	28		
\adl@currentcolumn			
.....	16, 19, 22, 23, 30–32, 36, 38, 57		
\adl@currentcolumnsave	16, 19, 38		
		<b>D</b>	
		\adl@dash	17, 38, 42
		\adl@dashcolor	28
		\dashgapcolor	8, 18, 64
		\adl@dashgapcolor@save	62
		\adl@dashgapcolor	22, 26, 61, 64, 66
		\dashlinedash	4, 14, 34
		\dashlinegap	4, 14, 34
		\adl@def@extract	66
		\adl@defcolor	64
		\adl@defflhdl	36
		\adl@depth	16, 19, 21, 32, 36, 38, 50
		\adl@depthsave	16, 19, 38
		\adl@discard	19, 32, 49, 53, 58
		\documentstyle	3
		\ifadl@doublerule	14, 38
		\doublerulesepcolor	7, 18, 64
		\adl@draw	35, 42, 44, 64
		\adl@drawi	44
		\adl@drawii	44
		\adl@drawiii	44
		\adl@drawvl	36, 42, 57
		<b>E</b>	
		\EAC	7
		\@elt	19, 40
		\ENAC	7
		\endArray	46, 62
		\endarray	36, 62
		\adl@endarray	20, 21, 36, 47
		\endfirsthead	51
		\endfoot	51
		\endhead	51
		\endlastfoot	51
		\endLongtable	46
		\endlongtable	49, 50
		\adl@endmakevlr	40
		\adl@endmakevlrconn	40
		\adl@endmakevlrcut	40
		\adl@endmbox	22, 24, 29, 49
		\adl@@endmbox	49
		\@endpbox	20, 21, 29, 47, 49, 61
		\@@endpbox	20, 21, 29, 47, 49, 54
		\endTabular	46
		\endtabular	36
		\endTabular*	46
		\endtabular*	36
		environments:	
		longtable	19, 48
		Array	7



\adl@LTheadR	57	\adl@org@@startpbox	17, 20
\adl@LThtfsave	51	\adl@org@activate	61
\adl@LTihdashline	49, 52, 67	\adl@org@arrayclassz	17, 20
\adl@LTiisplit	58	\adl@org@classz	17, 20, 65
\adl@LTinactivate	49	\adl@org@ccline	17, 20, 33, 35
\adl@LTinactivehdl	49, 52, 67	\adl@org@endlongtable	49, 50
\adl@LTinit	57	\adl@org@endpbox	17, 20
\adl@LTisplit	58	\adl@org@inactivate	61
\adl@LTixhline	53, 67	\adl@org@LT@echunk	49, 50
\adl@LTkill	52	\adl@org@LT@end@hd@ft	49, 51
\adl@LTkillend	52	\adl@org@LT@kill	49, 52
\adl@LTlastrow	50	\adl@org@LT@make@row	49
\adl@LTpagetotal	48, 54, 57	\adl@org@LT@output	49, 55
\adl@LTrowdiscard	58	\adl@org@tabclassz	17, 20
\adl@LTrowrelax	58	\output	48
\adl@LTsplit	58		
\adl@LTsplitend	58	P	
\adl@LTstartpbox	49, 54	packages:	
\adl@LTth	51	array	3, 7
\adl@LTxhline	53	colortab	3, 7
\adl@LTxsplitleft	58	colortbl	3, 7
\adl@LTysplitleft	58	longtable	3, 8, 48
		\PackageWarning	46
M		\adl@paintdashgap	64
\adl@makevrl	40	\adl@preamble	29
\adl@makevrlL	36, 38, 57	\adl@preamininit	22, 27, 28
\adl@makevrlR	36, 38	\adl@putlrc	24, 25, 26
\adl@mcaddamp	29		
\adl@mcargarraydashrule	30, 31	R	
\adl@mcarrraydashrule	30, 31	\rowcolor	7
\adl@mcarrrayrule	30, 31	\adl@rows	36, 38, 57, 58
\@medpenalty	53	\adl@rowsL	19, 32, 34, 36, 38, 48, 51, 57
\MessageBreak	46	\adl@rowsLsave	19, 38
\@mkpream	22	\adl@rowsR	19, 32, 34, 36, 38, 48, 51, 57
\adl@mkpream	22	\adl@rowsRsave	19, 38
\multicolumn	4, 29, 62		
		S	
N		\setcolor	42
\NAC	7	\adl@setcolor	28
\adl@ncol	16, 23, 26, 49	\adl@startmbox	22, 24, 29
\newcolumnntype	7	\@startpbox	29, 49, 54
\@nextchar	26	\@@startpbox	20, 21, 29, 47, 49, 54
\adl@noalign	17, 64		
\adl@nocolor	28	T	
\nodashgapcolor	8, 64	\@tabarray	18
\@notdefinable	45	\@tabclassz	20, 21, 25, 47
\adl@notdefinable	45	\Tabular	46
O		Tabular (environment)	7
\adl@org@@endpbox	17, 20	\tabular	45
		tabular (environment)	4



<code>\Tabular*</code> .....	<u>46</u>	<code>\adl@vlineR</code> .....	20, 21, <u>28</u> , 47
<code>\tabular*</code> .....	45	<code>\adl@vrow</code> .....	<u>36</u> , 38, 42, 57
<code>\adl@Tabular</code> .....	<u>45</u>	<code>\adl@vrowL</code> .....	<u>36</u> , 38, 42, 57
<code>\adl@Tabularstar</code> .....	<u>45</u>	<code>\adl@vrowR</code> .....	<u>36</u> , 38, 57
<code>\@tempcnta</code> .....	17	<code>\adl@vmakevlr</code> .....	40
<code>\@tempcntb</code> .....	17	<code>\adl@vrule</code> .....	35, <u>43</u>
<code>\@tempdima</code> .....	16	<b>W</b>	
<code>\@tempdimb</code> .....	16	<code>\@warning</code> .....	46
<code>\@tempswa</code> .....	15	<b>X</b>	
<code>\@testpach</code> .....	24, <u>25</u>	<code>\@xargarraycr</code> .....	<u>31</u>
<code>\adl@totalheight</code> ....	<u>16</u> , 19, 32, 34, 38, 51	<code>\@xarraycr</code> .....	<u>31</u>
<code>\adl@totalheightsave</code> .....	<u>17</u> , 19, 38	<code>\adl@xarraydashrule</code> .....	<u>26</u>
<b>U</b>		<code>\@xhline</code> .....	34
<code>\usepackage</code> .....	3	<code>\adl@xhline</code> .....	33, <u>34</u>
<code>\ifadl@usingarypkg</code> .....	<u>15</u> , 23, 30, 31	<code>\xleaders</code> .....	44
<b>V</b>		<code>\@xtabularcr</code> .....	<u>31</u>
<code>\adl@vl@leftskip</code> .....	<u>42</u>	<b>Y</b>	
<code>\adl@vl@rightskip</code> .....	<u>42</u>	<code>\@yargarraycr</code> .....	<u>31</u>
<code>\adl@vl</code> .....	42, 57	<b>Z</b>	
<code>\adl@vlineL</code> .....	<u>42</u> , 57	<code>\ifadl@zwrule</code> .....	<u>15</u>
<code>\adl@vlineL</code> .....	20, 21, <u>22</u> , 26, <u>27</u> , <u>28</u>	<code>\ifadl@zwvrule</code> .....	<u>14</u> , 29
<code>\adl@vlineL</code> .....	20, 21, <u>28</u> , 47		
<code>\adl@vlineR</code> .....	20, 21, <u>22</u> , 26, <u>27</u> , <u>28</u>		

## Change History

v1.0	
General: The style was born on a good day ... (1993/04/01)	1
v1.05	
General: Cope with <code>\\</code> with negative optional vertical space. (1993/06/18)	1
v1.1	
General: Save and restore the <code>\catcode</code> for ‘@’. (1993/06/24)	1
v1.2	
General: Various changes shown below. (1998/07/16)	1
v1.2-1	
General: Add this document.	1
v1.2-2	
General: Cope with $\text{\LaTeX} 2\epsilon$ .	1
v1.2-3	
General: Allow mixture of vertical solid- and dash-lines.	1
v1.2-4	
General: Add the feature of explicit dash/gap specification.	1
v1.2-5	
General: Fix some bugs and change codes.	1
v1.3	
General: Fix one bug shown below. (1998/10/08)	1
<code>\adl@activatepbox</code> : <code>\def</code> -s for <code>\adl@mcarrayrule</code> etc. are enclosed in a group.	29
v1.4	
General: Make compatible with array package and add new features. (1999/06/25)	1
v1.4-1	
General: The following are changes of this document.	1
General: The history on the compatibility with <code>array</code> package.	3
General: Explanation of package loading is added.	3
General: Description of <code>\first/lasthdashline</code> is added.	4
General: Description of the real width of vertical lines is added.	5
General: Description of drawing mode is added.	5
General: Description of (in)activation is added.	6
General: Description of characters and commands of <code>array</code> package is added.	7
General: Description about ‘!’ of array package is added.	9
General: Reference to the section for drawing mode is added.	9
General: Description on minimum length is added.	9
General: Reference to the performance tuning section is added.	9
General: The title of section 4.1 is changed.	10
General: <code>\hfil</code> is replaced with <code>\hss</code> taking the possibility of negative wide columns into account.	10
General: Section 4.12 is added.	43
General: Section 4.13 is added.	45
General: Thank to more people.	67
v1.4-2-1	
General: The following are for the general compatibility with <code>array</code> .	1
<code>\ifadl@usingarypkg</code> : Introduced to know if <code>array</code> is loaded.	15
<code>\adl@ncol</code> : Introduced for new column counting in preamble construction.	16
<code>\adl@everyvbox</code> : Introduced for a tricky modification of <code>\@array</code> .	17
<code>\adl@array</code> : Introduced to save original definition of <code>\@array</code> .	17

<code>\@array</code> : Drastically modified to avoid copy-and-modify. . . . .	17
<code>\@@array</code> : Introduced because <code>array</code> uses it. . . . .	18
<code>\adl@arrayinit</code> : Modified for new column counting in preamble construction. . . . .	19
<code>\@mkpream</code> : Modified for new column counting and control sequence redefinition. . . . .	22
<code>\@addamp</code> : Modified for new column counting in preamble construction. . . . .	23
<code>\@testpach</code> : The version for <code>array</code> is introduced. . . . .	24
<code>\@classz</code> : Introduced because <code>array</code> uses it. . . . .	24
<code>\adl@class@start</code> : Introduced for class number identification. . . . .	25
<code>\adl@class@iiiorvii</code> : Introduced for class number identification. . . . .	25
<code>\adl@class@start</code> : Introduced for class number identification. . . . .	26
<code>\adl@class@iiiorvii</code> : Introduced for class number identification. . . . .	26
<code>\adl@arrayrule</code> : Modified to replace <code>\adl@columns</code> with <code>\adl@ncol</code> . . . . .	26
<code>\adl@arraydashrule</code> : Modified to replace <code>\adl@columns</code> with <code>\adl@ncol</code> . . . . .	26
<code>\adl@argarraydashrule</code> : Modified to replace <code>\adl@columns</code> with <code>\adl@ncol</code> . . . . .	26
<code>\adl@xarraydashrule</code> : Modified to pretend <code>p</code> or <code>@</code> depending on if <code>array</code> is in use. . . . .	26
<code>\adl@xarraydashrule</code> : Modified to refer <code>\adl@class@start</code> rather than L <sup>A</sup> T <sub>E</sub> X's 6. . . . .	26
<code>\adl@colhtdp</code> : Initialized by calling <code>\adl@preaminit</code> . . . . .	28
<code>\adl@vlineL</code> : Initialized by calling <code>\adl@preaminit</code> . . . . .	28
<code>\adl@vlineR</code> : Initialized by calling <code>\adl@preaminit</code> . . . . .	28
<code>\@endpbox</code> : Introduced because <code>array</code> uses it. . . . .	29
<code>\multicolumn</code> : Modified for several reason. . . . .	29
<code>\adl@mcaddamp</code> : Introduced for the complaint on multiple columns if with <code>array</code> . . . . .	29
<code>\adl@activatepbox</code> : Introduced to do nothing if with <code>array</code> . . . . .	29
<code>\adl@mcargarraydashrule</code> : Modified to pretend <code>p</code> or <code>@</code> depending on if <code>array</code> is in use. . . . .	31
<code>\@xarraycr</code> : The version for <code>array</code> is introduced. . . . .	31
v1.4-2-2	
General: The following are to control the effective width of vertical lines. . . . .	1
<code>\ifadl@zwvrule</code> : Introduced to indicate vertical lines have null width. . . . .	14
<code>\ADLnullwidehline</code> : Introduced to make vertical lines null wide. . . . .	15
<code>\ADLsomewidehline</code> : Introduced to make vertical lines <code>\arraydashline</code> wide. . . . .	15
<code>\adl@xarraydashrule</code> : Modified to add invisible rule of <code>\arrayrulewidth</code> wide if <code>\ADLsome</code> wide. . . . .	26
<code>\adl@@vl</code> : Modified to make vertical line null wide only if <code>\ADLnullwide</code> . . . . .	42
v1.4-2-3	
General: The following are for inactivation of dash-line functions. . . . .	1
<code>\ifadl@inactive</code> : Introduced to indicate dash-line functions are inactive. . . . .	15
<code>\adl@org@arrayclassz</code> : Introduced to restore <code>\@arrayclassz</code> . . . . .	17
<code>\adl@org@tabclassz</code> : Introduced to restore <code>\@tabclassz</code> . . . . .	17
<code>\adl@org@classz</code> : Introduced to restore <code>\@classz</code> . . . . .	17
<code>\adl@org@@startpbox</code> : Introduced to restore <code>\@@startpbox</code> . . . . .	17
<code>\adl@org@@endpbox</code> : Introduced to restore <code>\@@endpbox</code> . . . . .	17
<code>\adl@org@endpbox</code> : Introduced to restore <code>\@endpbox</code> . . . . .	17
<code>\adl@org@ccline</code> : Introduced to restore <code>\ccline</code> . . . . .	17
<code>\adl@arrayinit</code> : Modified to call <code>\adl@inactivate</code> . . . . .	19
<code>\adl@inactivate</code> : Introduced to inactivate <code>\@arrayclassz</code> etc. . . . .	20
<code>\adl@inactivevl</code> : Introduced to emulate <code>'</code> and <code>;</code> by <code> </code> . . . . .	29
<code>\adl@inactivehdl</code> : Introduced to emulate <code>\hdashline</code> by <code>\hline</code> . . . . .	34
<code>\adl@inactivecdl</code> : Introduced to emulate <code>\cdashline</code> by <code>\ccline</code> . . . . .	35
<code>\adl@Array</code> : Introduced as the body of <code>\Array</code> . . . . .	45
<code>\adl@Tabular</code> : Introduced as the body of <code>\Tabular</code> . . . . .	45

\adl@Tabularstar:	Introduced as the body of \Tabular*.	45
\adl@notdefinable:	Introduced to check if \Array etc. are definable.	45
\Array:	Introduced as the always-active \array.	46
\Tabular:	Introduced as the always-active \tabular.	46
\Tabular*:	Introduced as the always-active \tabular*.	46
\endArray:	Introduced to \end the environment Array.	46
\endTabular:	Introduced to \end the environment Tabular.	46
\endTabular*:	Introduced to \end the environment Tabular*.	46
\ADLnoshorthand:	Introduced to nullify macros for shorthand activation.	46
v1.4-2-4		
General:	The following are for drawing mode to cope with the bug of \xleaders.	1
\adl@hcline:	Modified to use \adl@draw.	35
\adl@vline:	Modified to use \adl@draw.	42
\adl@hrule:	Introduced to draw a dash for horizontal lines in \adl@draw.	43
\adl@vrule:	Introduced to draw a dash for vertical lines in \adl@draw.	43
\adl@drawi:	Introduced as \adl@draw in mode 1.	44
\adl@drawii:	Introduced as \adl@draw in mode 2.	44
\adl@drawiii:	Introduced as \adl@draw in mode 3.	44
\adl@draw:	Introduced as the mode and axis independent line drawing macro.	44
\ADLdrawingmode:	Introduced to specify drawing mode.	45
v1.4-2-5		
General:	The following are to implement dashed version of \firsthline and \lasthline of array.	1
\hdashline:	Modified to make \adl@hdashline usable for \first/lasthdashline.	34
\adl@hdashline:	Modified to be usable for \first/lasthdashline.	34
\adl@ihdashline:	Introduced as the substitute of old \adl@hdashline.	34
\firsthdashline:	Introduced as the dashed version of \firsthline.	35
\lasthdashline:	Introduced as the dashed version of \lasthline.	35
\adl@defflhd1:	Introduced for the tricky definition of \adl@first/lasthdashline.	36
\adl@idefflhd1:	Introduced for the tricky definition of \adl@first/lasthdashline.	36
\adl@firsthdashline:	Introduced as the body of \firsthdashline.	36
\adl@lasthdashline:	Introduced as the body of \lasthdashline.	36
v1.4-2-6		
General:	The following are to fix the bug by which the depth of array/tabular was always zero.	1
\adl@finaldepth:	Introduced to measure the depth of the last row.	16
\adl@org@ccline:	Introduced to refer original version in modified \ccline.	17
\adl@ocr:	Modified to set \adl@finaldepth.	32
\hline:	Modified to set \adl@finaldepth to zero.	33
\ccline:	Modified to set \adl@finaldepth to zero.	33
\adl@endarray:	Modified to set the depth of array/tabular to \adl@finaldepth.	36
v1.4-2-7		
General:	The following are to rename macros for \cdashline.	1
\cdashline:	Modified to call renamed \adl@cdline.	35
\adl@cdline:	Renamed and modified to call renamed \adl@cdlinea/b.	35
\adl@cdlinea:	Renamed.	35
\adl@cdlineb:	Renamed.	35
v1.4-2-8		
General:	The following are to cope with very narrow or negative wide columns.	1

\adl@makev1rL: Modified to replace \hfil with \hss to prevent drawing vertical lines widen columns. ....	38
\adl@makev1rR: Modified to replace \hfil with \hss to prevent drawing vertical lines widen columns. ....	38
v1.4-2-9	
\adl@arrayinit: The bug of saving \adl@colsR is fixed. ....	19
v1.4-3	
General: Released to CTAN on 2000/07/04. ....	1
v1.5	
General: Make compatible with colortab, and fix bugs. (2000/07/12) ....	1
v1.5-1	
General: The following are for the compatibility with colortab. ....	1
General: The history on the compatibility with colortab package. ....	3
General: Caution about loading order of colortab is added. ....	3
General: Section 2.7 is added. ....	7
General: Description of colortab commands is added. ....	7
General: Caution about \AC/\EAC pair for vertical line coloring is added. ....	9
\adl@arrayinit: Use new macro \adl@arraysave to save registers/structures. ....	19
\adl@arraysave: Introduced to use in modified \CC@ of colortab. ....	19
\CC@: Modified to save/restore globals before/after height measurement. ....	47
v1.5-2	
General: The following are for bug fix of \adl@putlrc. ....	1
\adl@colhtdp: The pseudo-formal description of <put-lrc> is modified. ....	21
\adl@putlrc: \adl@putlrc must do \unhbox\adl@box to make glues effective. ....	26
v1.5-3	
General: The following are for bug fix of \adl@inactivate. ....	1
\adl@noalign: Move \adl@inactivate to \@array from \adl@arrayinit. ....	18
\adl@arrayinit: Move \adl@inactivate from \adl@arrayinit to \@array. ....	19
\adl@inactivate: Change \adl@inactivate caller to \@array. ....	20
General: Thank to Yaxin Liu. ....	67
v1.54	
General: Bug fixes. (2003/08/25) ....	1
v1.54-1	
General: The following are for bug fix of \adl@@v1. ....	1
\adl@v1row: Rows for vertical lines are replaced by \adl@drawv1. ....	37
\adl@drawv1: Introduced to draw vertical lines correctly if \ADLsomewide. ....	42
\adl@@v1: Insert a negative skip to left/right of the line if \ADLsomewide. ....	42
v1.54-2	
General: The following are for bug fix of activation. ....	1
\adl@noalign: Invoke \adl@activate if not \ifadl@inactive. ....	18
\adl@inactivate: Add \adl@argcr to inactivation. ....	20
\adl@activate: Introduced to activate \@arrayclassz etc. again. ....	20
\adl@act@arrayclassz: Introduced to activate \@arrayclassz etc. again. ....	47
v1.54-3	
General: The following are miscellaneous modifications. ....	1
\adl@hcline: Omit \vskip if the space is 0. ....	35
v1.6	
General: The following are for the compatibility with longtable. (2003/08/25) ....	1
General: The history on the compatibility with longtable package. ....	3
General: Caution about loading order of longtable is added. ....	3

General: Description of <code>longtable</code> is added. . . . .	8
General: Description of <code>discard</code> is added. . . . .	11
<code>\adl@discard</code> : Add initialization of <code>\adl@discard</code> . . . . .	19
General: Add a summary of activation/inactivation. . . . .	21
<code>\adl@cr</code> : Modified to insert <code>\adl@discard</code> . . . . .	32
<code>\adl@Longtable</code> : Introduced as the body of <code>\Longtable</code> . . . . .	45
<code>\Longtable</code> : Introduced as the always-active <code>\longtable</code> . . . . .	46
<code>\endLongtable</code> : Introduced to <code>\end</code> the environment <code>Longtable</code> . . . . .	46
<code>\ADLnoshorthanded</code> : <code>\Longtable</code> and <code>\endLongtable</code> are added. . . . .	46
General: §4.15 is added. . . . .	48
General: Thank to people for <code>longtable</code> . . . . .	67
v1.7	
General: The following are for the compatibility with <code>colortbl</code> . (2004/05/21) . . . . .	1
General: The history on the compatibility with <code>colortbl</code> package. . . . .	3
General: Caution about loading order of <code>colortbl</code> is added. . . . .	3
General: Description of <code>colortbl</code> and related commands is added. . . . .	7
General: Comment on vertical line coloring with <code>colortbl</code> is added. . . . .	9
General: Add notes for dash line coloring. . . . .	10
General: A dash/gap specification $d_j^i/g_j^i$ now has color. . . . .	11
<code>\endtabular</code> : Modified to refer proper <code>\endarray</code> depending on the existence of <code>colortbl</code> . . . . .	36
General: Codes for <code>longtable</code> is surrounded by <code>\ifx/\fi</code> . . . . .	48
General: §4.16 is added. . . . .	60
General: Thank to Klaus Dalinghaus and refer original <code>colortbl</code> . . . . .	67
v1.7-1	
General: The following are for null-wide horizontal lines. . . . .	1
<code>\ifadl@zwhrule</code> : Introduced to indicate horizontal lines have null width. . . . .	15
<code>\ADLnullwide</code> : Introduced to make horizontal lines null wide. . . . .	15
<code>\ADLsomewide</code> : Introduced to make horizontal lines <code>\arraydashline</code> wide. . . . .	15
<code>\adl@inactivate</code> : Remove <code>\cline</code> because our own version is needed for null-wide. . . . .	20
<code>\hline</code> : Modified to shift up if null-wide. . . . .	33
<code>\cline</code> : Modified to shift up if null-wide. . . . .	33
<code>\adl@hdashline</code> : Modified for null-wide horizontal lines. . . . .	34
<code>\adl@ihdashline</code> : <code>\adl@hline</code> is moved to <code>\adl@hdashline</code> for null-wide lines. . . . .	34
<code>\adl@inactivehdl</code> : Modified to shift up if null-wide. . . . .	34
<code>\adl@cdline</code> : Modified to shift up if null-wide. . . . .	35
<code>\adl@inactivecdl</code> : Modified to invoke <code>\cline</code> rather than <code>\adl@orgcline</code> for null-wide. . . . .	35
<code>\adl@hcline</code> : Modified not to shift null-wide <code>\cdashline</code> down. . . . .	35
<code>\adl@LThdashline</code> : Keep original without shift up because it is done by <code>\adl@LThdline</code> . . . . .	52
<code>\adl@LThdline</code> : Modified to shift up if null-wide. . . . .	53
v1.7-2	
General: The following are to fix the bug of <code>\arrayrulecolor</code> etc. in <code>colortbl</code> . . . . .	1
<code>\adl@noalign</code> : Introduced to fix a bug of <code>colortbl</code> . . . . .	17
<code>\adl@noalign</code> : Make <code>\adl@noalign \let-equal</code> to <code>\noalign</code> . . . . .	18
v1.7-3	
General: The following are for vertical line coloring. . . . .	1
<code>\adl@xarraydashrule</code> : Modified to add color arguments to <code>\adl@vlineL/R</code> . . . . .	26
<code>\adl@@vlineL</code> : Color arguments are added. . . . .	28
<code>\adl@@vlineR</code> : Color arguments are added. . . . .	28
<code>\adl@@ivline</code> : Invocations of <code>\adl@setcolor</code> are added. . . . .	28

<code>\adl@setcolor</code> : Introduced to color vertical lines. . . . .	28
<code>\adl@nocolor</code> : Introduced to examine if coloring is specified. . . . .	28
<code>\adl@dashcolor</code> : Introduced as the temporary variable of color specification of dashes. . .	28
<code>\adl@gapcolor</code> : Introduced as the temporary variable of color specification of gaps. . . .	28
<code>\adl@inactivevl</code> : Modified to color the <code>\vline</code> by the first argument. . . . .	29
<code>\adl@makevlr</code> : Modified to initialize <code>\adl@dashcolor</code> and <code>\adl@gapcolor</code> . . . . .	40
<code>\adl@iiimakevlr</code> : Modified to check color identity. . . . .	40
<code>\adl@ivmakevlr</code> : Modified not to see $d$ and $g$ which now have colors. . . . .	40
<code>\adl@addv1L</code> : Modified to add colors to $\delta$ and $\xi$ . . . . .	42
<code>\adl@addv1R</code> : Modified to add colors to $\delta$ and $\xi$ . . . . .	42
<code>\adl@@vl</code> : Modified to color dashes and gaps. . . . .	42
v1.71	
General: The following are for bug fix for <code>array</code> 's <code>m</code> -columns. (2004/7/31) . . . . .	1
<code>\@mkpream</code> : Modified to nullify <code>\adl@startmbox</code> and <code>\adl@endmbox</code> for <code>array</code> 's <code>m</code> -columns. .	22
<code>\@classz</code> : Modified to call <code>\adl@startmbox</code> and <code>\adl@endmbox</code> for <code>array</code> 's <code>m</code> -columns. . .	24
<code>\adl@startmbox</code> : Introduced to the bug fix of <code>array</code> 's <code>m</code> -columns. . . . .	29
<code>\adl@endmbox</code> : Introduced to the bug fix of <code>array</code> 's <code>m</code> -columns. . . . .	29
General: Thank to Morten Høgholm. . . . .	67
v1.72	
General: Bug fix and revision of §2.4. (2016/03/19) . . . . .	1
v1.72-1	
General: The following are for bug fix for footnotes in <code>longtable</code> 's <code>m</code> -columns. . . . .	1
<code>\LT@make@row</code> : Modified to add <code>\let</code> -assignments to <code>\adl@@endmbox</code> and <code>\adl@endbmox</code> so that footnotes are correctly processed at the closing of a <code>m</code> -type column. . . . .	49
<code>\adl@LTendmbox</code> : Added to process footnotes in <code>m</code> -type columns appropriately. . . . .	54
General: Thank to Maïeul Rouquette. . . . .	67
v1.72-2	
General: Revise §2.4 reflecting the fix of <code>\xleaders</code> . . . . .	5
General: Remove the caution about the dash segment dropping. . . . .	9
General: Change the title of §4.2 and rephrase sentences according to the fix of <code>\xleader</code> 's problem. . . . .	13
v1.73	
General: Bug fix. (2016/04/28) . . . . .	1
General: Thank to Maïeul Rouquette again. . . . .	67
v1.73-1	
General: The following are to fix the problem that the top edge a vertical (dash-)line is at the bottom of a horizontal line rather than it top. . . . .	1
General: Add a paragraph describing the perfect contacts of vertical and horizontal lines. .	4
General: Add the definition of $\eta_l$ and addition/subtraction of it for $\tau_k$ and $\beta_k$ . . . . .	12
General: Add $\eta = \text{\adl@lastconn}$ , its initialization and updates, and the addition to $\tau$ . .	38
<code>\adl@makevlr</code> : Add $\eta = \text{\adl@lastconn} \leftarrow 0$ . . . . .	40
<code>\adl@iimakevlr</code> : Modify the definition of <code>\adl@connect</code> to pass $h$ to <code>\adl@@connect</code> . . .	40
<code>\adl@iiimakevlr</code> : Replace two occurrences of $\tau \leftarrow \beta$ with $\tau \leftarrow \beta + \eta$ and add $\eta \leftarrow 0$ , where $\eta = \text{\adl@lastconn}$ . . . . .	40
<code>\adl@endmakevlrcut</code> : Add $\eta = \text{\adl@lastconn} \leftarrow 0$ . . . . .	40
<code>\adl@@connect</code> : Add $\eta = \text{\adl@lastconn} \leftarrow h$ with the added argument $h$ . . . . .	40
v1.73-2	
General: The following are to fix the bug that <code>\hdashline</code> is not properly processed in a <code>array/tabular</code> environment if <code>longtable</code> is loaded. . . . .	1

<code>\LT@array</code> : Add <code>\let</code> -assignment of <code>\adl@LThdashline</code> to <code>\adl@hdashline</code> so that the <code>longtable</code> version of <code>\adl@hdashline</code> is effective only in <code>longtable</code> environment rather than globally. ....	49
<code>\adl@LThdashline</code> : Renamed from <code>\adl@hdashline</code> to make it effective only in <code>longtable</code> environments. ....	52
v1.74	
General: The following are to fix the bug in the <code>array</code> -compatible mechanism by which <code>delarray</code> did not work well. ....	1
General: Comment on <code>plextarydshln</code> is added. ....	9
<code>\@@array</code> : Make <code>\@@array \let</code> -equal to <code>\@array</code> only when it is made so by <code>array</code> and the equality is kept. ....	18
<code>\endarray</code> : Add conditional invocation of <code>\@arrayright</code> . ....	36
<code>\endarray</code> : Add conditional invocation of <code>\@arrayright</code> . ....	62
General: Thank to Hironobu Yamashita. ....	67