The arydshln package*

Hiroshi Nakashima
(Toyohashi Univ. of Tech.)

2003/08/25

Abstract

This file gives LaTeX's `array` and `tabular` environments the capability to draw horizontal/vertical dash-lines.

Contents

*This file has version number v1.6, last revised 2003/08/25.

1 Introduction

In January 1993, Weimin Zhang kindly posted a style hvdashln written by the author, which draws horizontal/vertical dash-lines in LaTeX'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 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 TeX 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 LaTeX Tools Bundle and had trouble because these styles were incompatible with each other. Therefore, the style became compatible with array and now has 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 Feburary 2001, Craig Leech reported another compatibility problem with longtable. Although the author promised that the problem would be attacked some day, the issue have left long time[1] until three other complaints. Then the author attacked the problem hoping it is the last compatible issue[2].

2 Usage

2.1 Loading Package

The package is usable to both LaTeX 2_ε and LaTeX-2.09 users with their standard package loading declaration. If you use LaTeX 2_ε, simply do the following.

```
\usepackage{arydshln}
```

If you still love LaTeX-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 LaTeX Tools Bundle, and colortab package is that arydshln has to be loaded *after* array, longtable and/or colortab done. That is, the following is correct but reversing the order of `\usepackage` will cause some mysterious error.

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

[1] Two years and a half! Sorry Craig.
[2] Well, never ending, probably.

2.2 Basic Usage

array
tabular

You can simply use `array` or `tabular(*)` environments with standard preamble, such as `{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
\cdashline

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

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

\firsthdashline
\lasthdashline

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

2.3 Style Parameters

\dashlinedash
\dashlinegap

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.

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;

`;{`⟨*dash*⟩`/`⟨*gap*⟩`}`

instead of ordinary '`:`' and will have a dash-line with dash segments of ⟨*dash*⟩ long separated by spaces of ⟨*gap*⟩.

\hdashline
\cdashline

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

`\hdashline[`⟨*dash*⟩`/`⟨*gap*⟩`]`
`\cdashline{`⟨*col1*⟩`-`⟨*col2*⟩`}[`⟨*dash*⟩`/`⟨*gap*⟩`]`

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` `\ADLsomewide`

The vertical solid and dashed lines are drawn as if their width is zero, as standard LaTeX'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 LaTeX 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 TeX 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 as the left and upper lines in Figure 1(1) are.

However, because of a bug (or buggy feature) of `\xleaders`, there is a small possibility that a dash segment near the right/bottom end drops as right and lower lines in (1) of the figure shows. To cope with this problem, you may change the *drawing mode* by `\ADLdrawingmode{`$\langle m \rangle$`}` as follows.

`\ADLdrawingmode`

- $m = 1$
  As shown in Figure 1(1), most beautiful in almost all cases as the left/upper lines, but has a small possibility to produce an ugly result as right/lower lines. 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.
- $m = 3$
  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.

It is recommended to use default mode 1 unless you have an ugly result in the final version of your manuscript, because the correctness of mode 1 is very sensitive to the length of dash-lines.

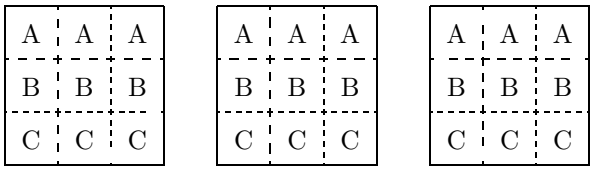

Figure 1: Drawing mode controlled by `\ADLdrawingmode`

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, 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 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` `Tabular` If you feel it tiresome to type the long command/environment name for the activation, 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 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`, `\firsthilne` 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 a ordinary table inactivating dash line functions by `\ADLinactivate`.

`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[3] 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.

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. As described in §2.5, dash-lines drawn in the default mode 1 may lack a dash segment near its right/bottom end.

[3] More precisely, drawn according to the column widths established by all the chunks preceding page output.

5. 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 $1.5 \times$ `\dashlinedash` $+$ `\dashlinegap`.

6. As described in §2.6, the processing speed for `array` and `tabular` environment will become slower even if dash-lines are not included.

7. 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.

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 TEX's sophisticated mechanism of the rule extension in the surrounding box. Since 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 describe later), drawing dash-lines in each row will not produce a good result. Look at the following two examples closely.

A	B
A	B

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 draw 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 hight and depth of two rows. In general, if we know the total hight 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.

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

1. The list of row information $R = \langle r_1, r_2, \ldots, r_N \rangle$.
2. The i^{th} element of R, r_i, is one of the following[4].
   (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. $C_i^L = \langle e_1^i, e_2^i, \ldots, e_m^i \rangle$ 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 of the segment. For a solid line segment, $d_j^i = g_j^i = 0$.

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.

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

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

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

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

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

[4] In the real implementation, the structure of r_i is slightly different.

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 $S_k = \{e^i_j, \ldots, e^{i'}_{j'}\}$ as follows.

- L_k is π^{th}_k line in the γ^{th}_k column where $\gamma_k = c^i_j = \ldots = c^{i'}_{j'}$ and $\pi_k = p^i_j = \ldots = p^{i'}_{j'}$.
- L_k has the dash size $\delta_k = d^i_j = \ldots = d^{i'}_{j'}$ and gap size $\xi_k = g^i_j = \ldots = g^{i'}_{j'}$.
- The top and bottom ends of L_k are at τ_k and β_k above the bottom of the `array`/`tabular`, where;

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

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

$$\sigma_1 L_1 \sigma_2 L_2 \ldots L_{n-1} \sigma_n L_n \sigma_{n+1} \texttt{\textbackslash cr}$$

where;

$$\sigma_1 = \texttt{\textbackslash omit}[\texttt{\textbackslash hss\&\textbackslash omit}]^{\gamma_1 - 1}$$

$$\sigma_{1<k\leq n} = \begin{cases} \texttt{\textbackslash null} & \text{if } \gamma_{k-1} = \gamma_k \wedge \pi_{k-1} = \pi_k \\ \texttt{\textbackslash hskip\textbackslash doublerulesep} & \text{if } \gamma_{k-1} = \gamma_k \wedge \pi_{k-1} \neq \pi_k \\ [\texttt{\textbackslash hss\&\textbackslash omit}]^{\gamma_k - \gamma_{k-1}} & \text{if } \gamma_{k-1} \neq \gamma_k \end{cases}$$

$$\sigma_{n+1} = [\texttt{\textbackslash hss\&\textbackslash omit}]^{\Gamma - \gamma_n - 1}\texttt{\textbackslash hss}.$$

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

Dash-lines at the right edges of columns are similarly drawn by processing C^R_i with the following modifications.

$$\begin{aligned} k < k' \leftrightarrow {}^{\forall} e^i_j \in S_k, \forall e^{i'}_{j'} \in S_{k'} ((c^i_j < c^{i'}_{j'}) \vee \\ (c^i_j = c^{i'}_{j'} \wedge p^i_j > p^{i'}_{j'}) \vee \\ (c^i_j = c^{i'}_{j'} \wedge p^i_j = p^{i'}_{j'} \wedge i < i')) \end{aligned}$$

$$\sigma_1 = \texttt{\textbackslash omit\textbackslash hss}[\texttt{\&\textbackslash omit\textbackslash hss}]^{\gamma_1 - 1}$$

$$\sigma_{k>1} = \begin{cases} \texttt{\textbackslash null} & \text{if } \gamma_{k-1} = \gamma_k \wedge \pi_{k-1} = \pi_k \\ \texttt{\textbackslash hskip\textbackslash doublerulesep} & \text{if } \gamma_{k-1} = \gamma_k \wedge \pi_{k-1} \neq \pi_k \\ [\texttt{\&\textbackslash omit\textbackslash hss}]^{\gamma_k - \gamma_{k-1}} & \text{if } \gamma_{k-1} \neq \gamma_k \end{cases}$$

$$\sigma_{n+1} = [\texttt{\&\textbackslash omit\textbackslash hss}]^{\Gamma - \gamma_n - 1}$$

4.2 Another Problem and Imperfect Solutions

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 $10\,\texttt{sp}$[5], which is unknown for us if horizontal but known for TEX, 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

[5]This small amount is added by `\xleaders` in order to, according to the comment in `tex.web`, compensate floating point rounding error.

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 ε. 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`.

$$\begin{aligned}
&D_0(d/2)G_0^l(g/2)\texttt{\textbackslash xleaders\textbackslash hbox\{}G^r(g/2)D(d)G^l(g/2)\texttt{\}\textbackslash hss}\ G_n^r(g)D_{n+1}(d/2)\\
&\quad = D_0(d/2)G_0^l(g/2)G_0^m(\varepsilon')\left(G_0^r(g/2)D_1(d)G_1^l(g/2)\right)G_1^m(\varepsilon)\\
&\qquad\qquad \left(G_1^r(g/2)D_2(d)G_2^l(g/2)\right)G_2^m(\varepsilon)\\
&\qquad\qquad \dots\\
&\qquad\qquad G_{n-1}^m(\varepsilon)\left(G_{n-1}^r(g/2)D_n(d)G_n^l(g/2)\right)G_n^m(\varepsilon')G_n^r(g/2)D_{n+1}(d/2)\\
&\quad = D_0(d/2)G_0(g+\varepsilon')D_1(d)G_1(g+\varepsilon)\dots G_{n-1}(g+\varepsilon)D_n(d)G_n(g+\varepsilon')D_{n+1}(d/2)
\end{aligned}$$

The problem is that ε' could be negative and TEX mistakingly ignores this possibility. That is, since TEX does not put `\hbox` beyond the right edge of `\xleaders`, the rightmost `\hbox` is omitted if ε' is negative as described in §2.5.

Since it is (almost) impossible to know the length of a horizontal line, we cannot cope with this problem by adding or subtracting its length. Thus we introduced *drawing mode* to have imperfect solutions. In the mode 2, we draw a line by the following sequence.

$$\begin{aligned}
&D_0(d/2)G_0^l(g/2)G_{0'}^r(g/2)D_{1'}(d)G_{1'}^l(g/2)G(-d-g)\\
&\quad \texttt{\textbackslash xleaders\textbackslash hbox\{}G^r(g/2)D(d)G^l(g/2)\texttt{\}\textbackslash hss}\\
&\quad G(-d-g)G_{n'}^r(g/2)D_{n'}(d)G_{n'}^l(g/2)G_n^r(g)D_{n+1}(d/2)
\end{aligned}$$

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.

`\dashlinedash` `\dashlinegap` `\hdashlinewidth` `\hdashlinegap`

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

```
%% Register Declaration

```

```
\newdimen\dashlinedash \dashlinedash4pt %
\newdimen\dashlinegap \dashlinegap4pt %
\let\hdashlinewidth\dashlinedash
\let\hdashlinegap\dashlinegap

```

Next, the following six switches are declared.

`\ifadl@leftrule`
- `\ifadl@leftrule` is used in the preamble analysis macro `\@mkpream` and is true during it processes leading characters for solid- and dash-lines, i.e. '`|`', '`:`', and '`;`'.

`\ifadl@connected`
- `\ifadl@connected` is used to indicate the *connection* $e^i_j \sim e^{i'}_{j'}$. When we process $e^{i'}_{j'}$, the switch is true iff ${}^{\exists}e^i_j(e^i_j \sim e^{i'}_{j'})$.

`\ifadl@doublerule`
- `\ifadl@doublerule` is used to make σ_k. When we are to make $\sigma_k L_k$, it is true iff $\gamma_{k-1} = \gamma_k \wedge \pi_{k-1} = \pi_k$.

`\ifadl@zwvrule`
- `\ifadl@zwvrule` controls the *real* width of vertical lines. If it is true, lines are drawn as if their width is zero following LaTeX's standard. Otherwise, their width `\arrayrulewidth` contribute to the width of columns as array does.

`\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 modified. 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 `\@tempswa`.

```
\newif\ifadl@leftrule
\newif\ifadl@connected
\newif\ifadl@doublerule
\newif\ifadl@zwvrule
\newif\ifadl@usingarypkg
\ifx\extrarowheight\undefined \adl@usingarypkgfalse
\else                         \adl@usingarypkgtrue  \fi
\newif\ifadl@inactive \adl@inactivefalse

```

`\ADLnullwide` `\ADLsomewide` The switch `\ifadl@zwvrule` is turned on/off by user interface macros `\ADLnullwide` and `\ADLsomewide`. Its initial value is the complement of `\adl@usingarypkg`.

`\ADLactivate` `\ADLinactivate` The switch `\ifadl@inactive` is also turned on/off by user interface macros `\ADLinactivate` and `\ADLactivate`.

```
\def\ADLnullwide{\adl@zwvruletrue}
\def\ADLsomewide{\adl@zwvrulefalse}
\ifadl@usingarypkg \ADLsomewide \else \ADLnullwide \fi

\def\ADLactivate{\adl@inactivefalse}
\def\ADLinactivate{\adl@inactivetrue}

```

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.

\adl@height
\adl@depth
- The \dimen registers \adl@height and \adl@depth 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 \adl@box and are updated if they are less.

\adl@heightsave
\adl@depthsave
Since we have to update these register \global-ly to pass their value across & and we may have a column containing array/tabular, they are saved into \adl@heightsave/\adl@depthsave at the beginning of the environment and are restored at its end.

\adl@finaldepth
The other \dimen register \adl@finaldepth 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 array/tabular down because we add extra two \smash-ed rows which make the depth of array/tabular zero.

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

```
\newbox\adl@box
\newdimen\adl@height \newdimen\adl@heightsave
\newdimen\adl@depth \newdimen\adl@depthsave
\newdimen\adl@finaldepth
```

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

\adl@columns
\adl@ncol
- \adl@columns has the number of columns specified in the preamble of the environment. Because of a complicated reason related to the compatibility with array, we cannot count up \adl@columns directly but increment \adl@ncol when each column of preamble is built and move its value to \adl@columns after the preamble is constructed.

\adl@currentcolumn
\adl@currentcolumnsave
- To process \multicolumn, we have to know the column number where it appears. Thus we have a column counter \adl@currentcolumn which is \global-ly incremented when each column is built. Because of the \global assignment, the counter has to be saved/restored into/from \adl@currentcolumnsave.

\adl@totalheight
- In the real implementation, τ_k and β_k are calculated by the following equations rather than those shown in §4.1.

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

\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\dimen$n$ which has the height plus depth of i^{th} row, h_i will be an approximation of \dimen$n$ 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\dimen$n$. Therefore, h_i is an integer and thus \adl@totalheight is too.

\adl@totalheightsave

Because of the \global assignment to \adl@totalheight to pass its value across rows, it has to be saved/restored into/from \adl@totalheightsave.

\adl@dash
\adl@gap

- In order to check $e^i_j \sim e^{i'}_{j'}$, d^i_j and g^i_j are kept in the registers \adl@dash and \adl@gap when we process $e^{i'}_{j'}$. As explained above, d^i_j and g^i_j are integers and thus \adl@dash and \adl@gap are \count registers.

\adl@cla
\adl@clb

- The coding of \cdashline is similar to that of \cline in LaTeX-2.09 which uses two global \count registers \@cla and \@clb. These registers are omitted from LaTeX 2_ε because its \cline is completely recoded. We could adopt new coding but it requires some other macro definitions that LaTeX-2.09 does not have. Thus we simply introduce new global counters \adl@cla and \adl@clb for \cdashline in order to make \cdashline work in both LaTeX-2.09 and LaTeX 2_ε.

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

```
\newcount\adl@columns \newcount\adl@ncol
\newcount\adl@currentcolumn \newcount\adl@currentcolumnsave
\newcount\adl@totalheight \newcount\adl@totalheightsave
\newcount\adl@dash \newcount\adl@gap
\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 LaTeX and array, we need to use \everyvbox in our own definition of \@array. The register is used to save the contents of \everyvbox.

```
\newtoks\adl@everyvbox

```

\adl@org@arrayclassz
\adl@org@tabclassz
\adl@org@classz
\adl@org@@startpbox
\adl@org@@endpbox
\adl@org@endpbox
\adl@org@cline

The other declarative stuffs are the sequence of \let 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 \adl@org@cline is also referred in its modified version.

```
\let\adl@org@arrayclassz\@arrayclassz
\let\adl@org@tabclassz\@tabclassz
\let\adl@org@classz\@classz
\let\adl@org@@startpbox\@@startpbox
\let\adl@org@@endpbox\@@endpbox
\let\adl@org@endpbox\@endpbox
\let\adl@org@cline\cline

%%^L
```

4.4 Initialization

\adl@array
\@array

LaTeX's macro \@array is modified to save and initialize registers and data structures which are \global-ly updated in order to allow nested array/tabular. This saving and initializing are performed by \adl@arrayinit as explained below. The problem in the modification is that the code of \@array in array is completely different from that of LaTeX original.

The main difference is that LaTeX 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 after \@preamble is thrown into TEX's *stomach*. In our implementation, unfortunately, the number of columns has to be counted in \@mkpream and will be referred 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 LATEX 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 work 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.

Finally, 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.

\@@array Another stuff for the compatibility with array is to \let a control sequence \@@array be equal to \@array because it is referred in \@tabarray in array.

```
44
45 %% Initialization
46
47 \let\adl@array\@array \def\@array{\adl@everyvbox\everyvbox
48 \everyvbox{\adl@arrayinit \the\adl@everyvbox \everyvbox\adl@everyvbox}%
49 \ifadl@inactive \adl@inactivate \else \adl@activate \fi \adl@array}
50 \let\@@array\@array
51
```

\adl@arrayinit \adl@arraysave As described in §4.3, registers updated \global-ly, which are \adl@height, \adl@depth, \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 \adl@rowsR \adl@rowsLsave \adl@rowsRsave \adl@colsL \adl@colsR \adl@colsLsave \adl@colsRsave

- In the real implementation, the data structure R is split into two lists;

$$\texttt{\adl@rowsL} = R^L = \langle\langle C_1^L, h_1\rangle, \ldots\rangle$$
$$\texttt{\adl@rowsR} = R^R = \langle\langle C_1^R, h_1\rangle, \ldots\rangle$$

  and they are saved into \adl@rowsLsave and \adl@rowsRsave.

- When the i^{th} row is building, C_i^L and C_i^R are constructed in the macros \adl@colsL and \adl@colsR. They are saved into \adl@colsLsave and \adl@colsRsave.

\adl@connect \adl@discard In the real implementation, e_j^i is represented by a control sequence \@elt, and $connect(i)$ by \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 usural `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 generated. Without array, on the other hand, this assignment has no effect but safe because it is included in a group of `\vbox` etc.

```
\def\adl@arrayinit{%
        \adl@arraysave
        \global\adl@height\z@ \global\adl@depth\z@
        \global\adl@currentcolumn\@ne \global\adl@totalheight\z@
        \gdef\adl@rowsL{}\gdef\adl@rowsR{}\gdef\adl@colsL{}\gdef\adl@colsR{}%
        \let\@elt\relax \let\adl@connect\relax \def\adl@discard{\adl@connect}%
        \adl@columns\adl@ncol}
\def\adl@arraysave{%
        \adl@heightsave\adl@height
        \adl@depthsave\adl@depth
        \adl@currentcolumnsave\adl@currentcolumn
        \adl@totalheightsave\adl@totalheight
        \let\adl@rowsLsave\adl@rowsL
        \let\adl@rowsRsave\adl@rowsR
        \let\adl@colsLsave\adl@colsL
        \let\adl@colsRsave\adl@colsR}

```

`\adl@inactivate` If `\ADLinactivate` has effect and thus `\ifadl@inactive` is true, the macro `\adl@inactivate` is called from `\@array`[6]. This `\let`-s the following control sequences be equal to their counterparts in LaTeX and/or array package.

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

Note that we have to inactivate both `\@@endpbox` for LaTeX 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 LaTeX does not define `\@classz` but refers it which is either `\@arrayclassz` or `\@tabclassz`. Yet another remark is that we have to conceal `\cr` for `\adl@cr`/`\adl@argcr` and `\crcr` for `\adl@endarray` by bracing them from TeX'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
```

[6] Before v1.53, `\adl@inactivate` was called from `\adl@arrayinit` and thus invokded *after* the preamble of array is built. This was incorrect of course and made inactive version of `p`, `m` and `b` produce nothing.

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

```
\def\adl@inactivate{%
        \let\@arrayclassz\adl@org@arrayclassz
        \let\@tabclassz\adl@org@tabclassz
        \ifadl@usingarypkg \let\@classz\adl@org@classz \fi
        \let\@@startpbox\adl@org@@startpbox
        \let\@@endpbox\adl@org@@endpbox
        \let\@endpbox\adl@org@endpbox
        \def\adl@cr{\cr}%
        \def\adl@argcr##1{\cr}%
        \let\cline\adl@org@cline
        \def\adl@endarray{\crcr}%
        \let\adl@hline\@gobbletwo
        \let\adl@ihdashline\adl@inactivehdl
        \let\adl@cdline\adl@inactivecdl
        \let\adl@@vlineL\adl@inactivevl
        \let\adl@@vlineR\adl@inactivevl
        \let\adl@vlineL\adl@inactivevl
        \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[7]. To do that, `\adl@activate` makes `\@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.

```
\def\adl@activate{%
        \let\@arrayclassz\adl@act@arrayclassz
        \let\@tabclassz\adl@act@tabclassz
        \ifadl@usingarypkg \let\@classz\adl@act@classz \fi
        \let\@@startpbox\adl@act@@startpbox
        \let\@@endpbox\adl@act@@endpbox
        \let\@endpbox\adl@act@endpbox
        \let\adl@cr\adl@act@cr
        \let\adl@argcr\adl@act@argcr
        \let\cline\adl@act@cline
        \let\adl@endarray\adl@act@endarray
        \let\adl@hline\adl@act@hline
        \let\adl@ihdashline\adl@act@ihdashline
        \let\adl@cdline\adl@act@cdline
        \let\adl@@vlineL\adl@act@@vlineL
        \let\adl@@vlineR\adl@act@@vlineR
        \let\adl@vlineL\adl@act@@vlineL
        \let\adl@vlineR\adl@act@@vlineR}

%%^L
```

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

[7] Before v1.54, an active array/tabular in an inactive parent was not activated.

Table 1: Active and Inactive Operations

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

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 ⟨*lrc*⟩.

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

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`
`\adl@vlineR`

- '`|`', '`:`' and `;{`⟨*dash*⟩`/`⟨*gap*⟩`}` are converted to the following ⟨*vline*⟩.

$$\begin{aligned}
\langle \textit{vline} \rangle &::= [\texttt{\textbackslash hskip\textbackslash doublerulesep}]\langle \textit{vline-LR} \rangle \\
\langle \textit{vline-LR} \rangle &::= \texttt{\textbackslash adl@vlineL\{}\langle c \rangle\texttt{\}\{}\langle d \rangle/\langle g \rangle\texttt{\}} \mid \\
&\quad\ \texttt{\textbackslash adl@vlineR\{}\langle c \rangle\texttt{\}\{}\langle d \rangle/\langle g \rangle\texttt{\}} \\
\langle d \rangle &::= 0 \mid && \ldots \text{ for '}\texttt{|}\text{'} \\
&\quad\ \texttt{\textbackslash dashlinedash} \mid && \ldots \text{ for '}\texttt{:}\text{'} \\
&\quad\ \langle \textit{dash} \rangle && \ldots \text{ for '}\texttt{;}\text{'} \\
\langle g \rangle &::= 0 \mid && \ldots \text{ for '}\texttt{|}\text{'} \\
&\quad\ \texttt{\textbackslash dashlinegap} \mid && \ldots \text{ for '}\texttt{:}\text{'} \\
&\quad\ \langle \textit{gap} \rangle && \ldots \text{ for '}\texttt{;}\text{'}
\end{aligned}$$

Note that ⟨*c*⟩ is the column number (leftmost is 1) where the character appears.

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 @{⟨*text*⟩} and !{⟨*text*⟩} 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 LaTeX original is done by (modified) \@@startpbox and \@@endpbox and thus the preamble for 'p' is not modified. In 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
\@mkpream

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

Giving them their own definition is done by \adl@preaminit 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@preaminit is properly called *after* this final *assignment* to make \@preamble.

With array, on the other hand, calling \adl@preaminit 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 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.

```
%% Making Preamble

\let\adl@mkpream\@mkpream
\def\@mkpream#1{\let\adl@colhtdp\relax
        \let\adl@vlineL\relax \let\adl@vlineR\relax
        \global\adl@ncol\@ne \adl@leftruletrue
        \adl@mkpream{#1}\adl@columns\adl@ncol \afterassignment\adl@preaminit}

```

\@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 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 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.

```
\def\@addamp{\if@firstamp\@firstampfalse \else
```

```
        \@addtopreamble{\global\advance\adl@currentcolumn\@ne &}%
        \global\advance\adl@ncol\@ne \fi}
```

Since the implementation of `\@testpach` and macros for class-0 characters (i.e. `l`, `r` and `c`) is completely different between 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` be the macro corresponding to each character.

```
\ifadl@usingarypkg
\def\@testpach{\@chclass
 \ifnum \@lastchclass=6 \@ne \@chnum \@ne \else
  \ifnum \@lastchclass=7 5 \else
   \ifnum \@lastchclass=8 \tw@ \else
    \ifnum \@lastchclass=9 \thr@@
   \else \z@
   \ifnum \@lastchclass = 10 \else
   \edef\@nextchar{\expandafter\string\@nextchar}%
   \@chnum
   \if \@nextchar c\z@ \else
    \if \@nextchar l\@ne \else
     \if \@nextchar r\tw@ \else
   \z@ \@chclass
   \if\@nextchar |\@ne \let\@arrayrule\adl@arrayrule \else
   \if\@nextchar :\@ne \let\@arrayrule\adl@arraydashrule \else
   \if\@nextchar ;\@ne \let\@arrayrule\adl@argarraydashrule \else
    \if \@nextchar !6 \else
     \if \@nextchar @7 \else
      \if \@nextchar <8 \else
       \if \@nextchar >9 \else
  10
  \@chnum
  \if \@nextchar m\thr@@\else
   \if \@nextchar p4 \else
    \if \@nextchar b5 \else
   \z@ \@chclass \z@ \@preamerr \z@ \fi \fi \fi \fi \fi \fi
   \fi \fi  \fi  \fi  \fi  \fi  \fi \fi \fi \fi \fi \fi}
```

`\@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.

```
\def\@classz{\@classx
   \@tempcnta \count@
```

```
  \prepnext@tok
  \@addtopreamble{\ifcase \@chnum
     \hfil
     \adl@putlrc{\d@llarbegin \insert@column \d@llarend}\hfil \or
     \hskip1sp\adl@putlrc{\d@llarbegin \insert@column \d@llarend}\hfil \or
     \hfil\hskip1sp\adl@putlrc{\d@llarbegin \insert@column \d@llarend}\or
  $\setbox\adl@box\vcenter\@startpbox{\@nextchar}\insert@column \@endpbox $\or
  \setbox\adl@box\vtop \@startpbox{\@nextchar}\insert@column \@endpbox \or
  \setbox\adl@box\vbox \@startpbox{\@nextchar}\insert@column \@endpbox
 \fi}\prepnext@tok}
```

`\adl@class@start` `\adl@class@iiiorvii` Another stuffs for compatibility are to refer the class number of the beginning of preamble which is different between LaTeX and array, and that of '`p`' or '`@`' to get the argument of '`;`' as explained later. In case with array, the former is class-4 and we use '`@`' (class-7) for the latter.

```
\def\adl@class@start{4}
\def\adl@class@iiiorvii{7}
```

Without array

`\@testpach` The reason why and how we modify `\@testpach` of LaTeX is same as those of array.

```
\else
\def\@testpach#1{\@chclass \ifnum \@lastchclass=\tw@ 4\relax \else
        \ifnum \@lastchclass=\thr@@ 5\relax \else
                \z@ \if #1c\@chnum \z@ \else
                    \if #1l\@chnum \@ne \else
                    \if #1r\@chnum \tw@ \else
                \@chclass
                    \if #1|\@ne \let\@arrayrule\adl@arrayrule \else
                    \if #1:\@ne \let\@arrayrule\adl@arraydashrule \else
                    \if #1;\@ne \let\@arrayrule\adl@argarraydashrule \else
                    \if #1@\tw@ \else
                    \if #1p\thr@@ \else \z@ \@preamerr 0\fi
        \fi  \fi  \fi  \fi  \fi  \fi  \fi  \fi  \fi}
```

`\@arrayclassz` `\@tabclassz` Since LaTeX 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.

```
\def\@arrayclassz{\ifcase \@lastchclass \@acolampacol \or \@ampacol \or
                \or \or \@addamp \or
                \@acolampacol \or \@firstampfalse \@acol \fi
        \edef\@preamble{\@preamble
                \ifcase \@chnum
                    \hfil\adl@putlrc{$\relax\@sharp$}\hfil
                \or \adl@putlrc{$\relax\@sharp$}\hfil
                \or \hfil\adl@putlrc{$\relax\@sharp$}\fi}}
\def\@tabclassz{\ifcase \@lastchclass \@acolampacol \or \@ampacol \or
                \or \or \@addamp \or
                \@acolampacol \or \@firstampfalse \@acol \fi
        \edef\@preamble{\@preamble
```

```
        \ifcase \@chnum
                    \hfil\adl@putlrc{\@sharp\unskip}\hfil
                \or \adl@putlrc{\@sharp\unskip}\hfil
                \or \hfil\hskip\z@ \adl@putlrc{\@sharp\unskip}\fi}}
```

`\adl@class@start` `\adl@class@iiiorvii`

In LaTeX, the beginning of preamble is class-6 and we use '`p`' (class-3) to get the argument of '`;`'.

```
\def\adl@class@start{6}
\def\adl@class@iiiorvii{3}
\fi

```

Hereafter, codes for LaTeX 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.

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

```

`\adl@arrayrule` `\adl@arraydashrule` `\adl@argarraydashrule` `\adl@xarraydashrule`

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{`$\langle c^L\rangle$`}{`$\langle c^R\rangle$`}{`$\langle d\rangle$`/`$\langle g\rangle$`}`

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{`$\langle c^L\rangle$`}{`$\langle d\rangle$`/`$\langle g\rangle$`}` 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{`$\langle c^R\rangle$`}{`$\langle d\rangle$`/`$\langle g\rangle$`}` 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.

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` `\adl@classvfordash`

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 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 set `\@chclass` to 1 to indicate that the last item is a vertical line specification.

```
\def\adl@arrayrule{%
        \adl@xarraydashrule
                {\@ne}{\adl@ncol}{{\z@/\z@}}}
\def\adl@arraydashrule{%
```

```
        \adl@xarraydashrule
                {\@ne}{\adl@ncol}%
                {{\dashlinedash/\dashlinegap}}}
\def\adl@argarraydashrule{%
        \adl@xarraydashrule
                {\@ne}{\adl@ncol}{}%
        \@chclass\adl@class@iiiorvii \let\@classv\adl@classvfordash}
\def\adl@xarraydashrule#1#2#3{%
        \ifnum\@lastchclass=\adl@class@start\else
        \ifnum\@lastchclass=\@ne\else
                \adl@leftrulefalse \fi\fi
        \ifadl@zwvrule\else \ifadl@inactive\else
                \@addtopreamble{\vrule\@width\arrayrulewidth
                        \@height\z@ \@depth\z@}\fi \fi
        \ifadl@leftrule
                \@addtopreamble{\adl@vlineL{\number#1}#3}%
        \else   \@addtopreamble{\adl@vlineR{\number#2}#3}\fi}
\let\adl@classv\@classv
\def\adl@classvfordash{\@addtopreamble{{\@nextchar}}\let\@classv\adl@classv
        \@chclass\@ne}

%%^L
```

4.6 Building Columns

`\adl@preaminit` `\adl@colhtdp` `\adl@vlineL` `\adl@vlineR`

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

```

%% Building Columns

\def\adl@preaminit{\let\adl@colhtdp\adl@@colhtdp
        \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.

```
\def\adl@@colhtdp{%
        \ifdim\adl@height<\ht\adl@box \global\adl@height\ht\adl@box \fi
        \ifdim\adl@depth<\dp\adl@box \global\adl@depth\dp\adl@box\fi}

```

`\adl@@vlineL` `\adl@@vlineR` `\adl@@ivline`

The macro `\adl@@vlineL`⟨c⟩`{`⟨d⟩`/`⟨g⟩`}` adds the element $e = \langle c, d, g\rangle$ = `\@elt{`⟨c⟩`}{`⟨d⟩`}{`⟨g⟩`}` to the tail of the list `\adl@colsL` to construct C_i^L. The macro `\add@@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 ⟨d⟩ and ⟨g⟩ are extracted by the macro `\adl@ivline` which converts given dimensional value of them to integers. It also set ⟨d⟩

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.

```
\def\adl@@vlineL#1#2{\adl@ivline#2\@nil
        \xdef\adl@colsL{\adl@colsL
                \@elt{#1}{\number\@tempcnta}{\number\@tempcntb}}}
\def\adl@@vlineR#1#2{\adl@ivline#2\@nil
        \xdef\adl@colsR{\@elt{#1}{\number\@tempcnta}{\number\@tempcntb}%
                \adl@colsR}}
\def\adl@ivline#1/#2\@nil{%
        \@tempdima#1\relax \@tempcnta\@tempdima
        \@tempdima#2\relax \@tempcntb\@tempdima
        \ifnum\@tempcnta>\z@ \else \@tempcnta\z@ \@tempcntb\z@ \fi
        \ifnum\@tempcntb>\z@ \else \@tempcnta\z@ \@tempcntb\z@ \fi}
```

`\adl@colhtdp` `\adl@vlineL` `\adl@vlineR`

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 case with array, `\adl@colhtdp` etc. are temporarily `\relax` when `\@preamble` is being generated in the group of `\@mkpream`, and regain their own definition outside the group where the completed `\@preamble` is referred.

```
\adl@preaminit
```

`\adl@inactivevl`

If `\ADLinactivate` is in effect, `\adl@vlineL/R` and `\adl@@vlineL/R` are `\let`-equal to `\adl@inactivevl`. This macro simply put a `\vrule` by `\vline` with/without negative `\hskip` of a half of `\arrayrulewidth` wide depending on `\ifadl@zwvrule`, discarding its arguments.

```
\def\adl@inactivevl#1#2{\ifadl@zwvrule \hskip-.5\arrayrulewidth \fi
        \vline \ifadl@zwvrule \hskip-.5\arrayrulewidth \fi}
```

`\@@startpbox` `\@@endpbox` `\@endpbox`

The macros to make `\parbox` for 'p' (and 'm' and 'b' of array), `\@@startpbox` and `\@@endpbox`, are modified for height/depth measurement. The code for `\@@endpbox` is based on that of LaTeX 2_ε to fix the bug of `\strut`-ing in LaTeX-2.09, but `\@finalstrut` is manually expanded because it is not available in LaTeX-2.09.

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.

```
\def\@@startpbox#1{\setbox\adl@box\vtop\bgroup \hsize#1\@arrayparboxrestore}
\def\@@endpbox{\unskip \ifhmode \nobreak
        \vrule\@width\z@\@height\z@\@depth\dp\@arstrutbox \fi
        \par \egroup \adl@colhtdp \box\adl@box \hfil}
\let\@endpbox\@@endpbox

%%^L
```

4.7 Multi-columns

`\multicolumn` `\adl@preamble` `\adl@mcaddamp` `\adl@activatepbox`

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

- 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 multicolumn 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 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@preaminit` called in `\@mkpream`, we call `\adl@preaminit` 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`-ed).

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

```
%% Multi-Columns

\def\multicolumn#1#2#3{\multispan{#1}\begingroup \begingroup
        \def\adl@arrayrule{\adl@mcarrayrule{#1}}%
        \def\adl@arraydashrule{\adl@mcarraydashrule{#1}}%
        \def\adl@argarraydashrule{\adl@mcargarraydashrule{#1}}%
        \let\@addamp\adl@mcaddamp
        \@mkpream{#2}\@addtopreamble\@empty
        \global\let\adl@preamble\@preamble \endgroup
        \let\@preamble\adl@preamble
        \def\@sharp{#3}\let\protect\relax
        \adl@activatepbox
        \adl@preaminit
        \@arstrut \@preamble\hbox{}\endgroup
        \global\advance\adl@currentcolumn#1\ignorespaces}
\ifadl@usingarypkg
        \def\adl@mcaddamp{\if@firstamp\@firstampfalse \else\@preamerror5\fi}
        \let\adl@activatepbox\relax
\else
```

```
        \let\adl@mcaddamp\@addamp
        \def\adl@activatepbox{\let\@startpbox\@@startpbox
                \let\@endpbox\@@endpbox}
\fi
```

\adl@mcarrayrule
\adl@mcarraydashrule
\adl@mcargarraydashrule

The preamble parts for vertical lines are constructed by the macros `\adl@mcarrayrule`, `\adl@mcarraydashrule` and `\adl@mcargarraydashrule` which are passed the first argument $\langle n \rangle$ of `\multicolumn` to know the number of columns to be `\span`-ed. 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;

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

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

```
\def\adl@mcarrayrule#1{\@tempcnta#1\advance\@tempcnta\adl@currentcolumn
        \advance\@tempcnta\m@ne
        \adl@xarraydashrule
                {\adl@currentcolumn}{\@tempcnta}{{\z@/\z@}}}
\def\adl@mcarraydashrule#1{\@tempcnta#1\advance\@tempcnta\adl@currentcolumn
        \advance\@tempcnta\m@ne
        \adl@xarraydashrule
                {\adl@currentcolumn}{\@tempcnta}%
                {{\dashlinedash/\dashlinegap}}}
\def\adl@mcargarraydashrule#1{\@tempcnta#1\advance\@tempcnta\adl@currentcolumn
        \advance\@tempcnta\m@ne
        \adl@xarraydashrule
                {\adl@currentcolumn}{\@tempcnta}{}%
        \@chclass\adl@class@iiiorvii \let\@classv\adl@classvfordash}

%%^L
```

4.8 End of Rows

\@xarraycr
\@xtabularcr
\@xargarraycr
\@yargarraycr

At the end of a i^{th} row, we have to calculate h_i which is the height plus depth of the 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 macros `\@xarraycr`, `\@xtabularcr`, `\@xargarraycr` are replaced with our own `\adl@cr`. The macro `\@yargarraycr`$\langle dimen \rangle$ is also modified but its `\cr` is replaced with `\adl@argcr`$\langle dimen \rangle$ to add (negative) `\dimen` to h_i. Note that `\@xargarraycr`$\langle dimen \rangle$ 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 LaTeX and array, we have to have two versions and choose one.

```

%% End of row

\ifadl@usingarypkg
\def\@xarraycr{\@ifnextchar[{\@argarraycr}{\ifnum0=`{}\fi\adl@cr}}
\else
\def\@xarraycr{\@ifnextchar[{\@argarraycr}{\ifnum0=`{\fi}$${}\adl@cr}}
\fi
```

```
\def\@xtabularcr{\@ifnextchar[{\@argtabularcr}{\ifnum0=`{\fi}\adl@cr}}
\def\@xargarraycr#1{\@tempdima#1\advance\@tempdima\dp\@arstrutbox
        \vrule\@height\z@\@depth\@tempdima\@width\z@
        \adl@cr}
\def\@yargarraycr#1{\adl@argcr{#1}\noalign{\vskip #1}}

```

`\adl@cr` `\adl@argcr` The macro `\adl@cr` and `\adl@argcr` perform `\cr` and then invoke the common macro `\adl@@cr`$\langle x\rangle$. 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`$\langle x\rangle$ 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`, η_s and δ_s. If so, the height and/or depth of the row are η_s and δ_s. Therefore, h_i is caluclated 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[8]. Thus the integer value of $h_i + x$ is `\global`-ly added to `\adl@totalheight`, and the elements $\langle C_i^L$=`\adl@colsL`$, h_i\rangle$ and $\langle C_i^R$=`\adl@colsR`$, h_i\rangle$ are added to the tail of R^L = `\adl@rowsL` and R^C = `\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*$\rangle$.

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

Since `\adl@discard` is `\def`-ined as `\adl@connect` by `\adl@arrayinit`, added `\adl@discard` transforms itself into `\adl@connect` if current envrionment 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 `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.

```
\def\adl@cr{\cr\noalign{\adl@@cr\z@}}
\def\adl@argcr#1{\cr\noalign{\adl@@cr{#1}}}
\def\adl@@cr#1{
        \ifdim\adl@height<\ht\@arstrutbox \adl@height\ht\@arstrutbox\fi
        \ifdim\adl@depth<\dp\@arstrutbox \adl@depth\dp\@arstrutbox\fi
        \advance\adl@height\adl@depth
        \global\advance\adl@totalheight\adl@height
```

[8] Before v1.54, negative $\langle x\rangle$ shrinks the hight 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`.

```
        \@tempdima#1\relax \global\advance\adl@totalheight\@tempdima
        \xdef\adl@rowsL{\adl@rowsL
                (\adl@colsL/\number\adl@height);%
                \ifdim#1=\z@\else (\adl@discard/\number\@tempdima);\fi}%
        \xdef\adl@rowsR{\adl@rowsR
                (\adl@colsR/\number\adl@height);%
                \ifdim#1=\z@\else (\adl@discard/\number\@tempdima);\fi}%
        \gdef\adl@colsL{}\gdef\adl@colsR{}
        \global\adl@currentcolumn\@ne
        \ifdim#1=\z@ \global\adl@finaldepth\adl@depth
        \else        \global\adl@finaldepth\z@\fi
        \global\adl@height\z@ \global\adl@depth\z@}

%%^L
```

4.9 Horizontal Lines

`\hline` `\cline`

The macro `\hline` is modified to add the element $connect(w)$ = `(\adl@connect/\number \arrayrulewidth)` to the end of R^L and R^R by `\adl@hline`, 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.

```

%% Horizontal Lines

\def\hline{\noalign{\ifnum0=`}\fi \hrule\@height\arrayrulewidth
        \adl@hline\adl@connect\arrayrulewidth
        \global\adl@finaldepth\z@
        \futurelet\@tempa\adl@xhline}
\def\cline{\noalign{\global\adl@finaldepth\z@}\adl@org@cline}

```

`\hdashline` `\adl@hdashline` `\adl@ihdashline`

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 `[`$\langle dash\rangle$`/`$\langle gap\rangle$`]`. Then the macro `\adl@ihdashline` adds $connect(w)$ to the end of $R^L$ and $R^R$, and 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`-ed 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.

```
\def\hdashline{\adl@hdashline\adl@ihdashline}
\def\adl@hdashline#1{\noalign{\ifnum0=`}\fi
        \@ifnextchar[%]
                        {#1}%
                        {#1[\dashlinedash/\dashlinegap]}}
\def\adl@ihdashline[#1/#2]{\adl@hline\adl@connect\arrayrulewidth
        \ifnum0=`{\fi}%
        \multispan{\adl@columns}\unskip \adl@hcline\z@[#1/#2]%
```

```
        \noalign{\ifnum0=‘}\fi
        \futurelet\@tempa\adl@xhline}
\def\adl@inactivehdl[#1/#2]{\hrule\@height\arrayrulewidth
        \futurelet\@tempa\adl@xhline}
```

`\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.

```
\def\adl@xhline{\ifx\@tempa\hline \adl@ixhline\fi
        \ifx\@tempa\hdashline \adl@ixhline\fi
        \ifnum0=‘{\fi}}
\def\adl@ixhline{\vskip\doublerulesep \adl@hline\relax\doublerulesep}
```

`\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).

```
\def\adl@hline#1#2{\@tempcnta#2
        \global\advance\adl@totalheight\@tempcnta
        \xdef\adl@rowsL{\adl@rowsL
                (#1/\number\@tempcnta);}%
        \xdef\adl@rowsR{\adl@rowsR
                (#1/\number\@tempcnta);}}
```

`\cdashline` `\adl@cdline` `\adl@cdlinea` `\adl@cdlineb` The macro `\cdashline` at first opens `\noalign` and then invokes `\adl@cdline` checking the existence of its optional argument `[`⟨*dash*⟩`/`⟨*gap*⟩`]`. The code of the macro `\adl@cdline` is based on that of `\@cline` in LaTeX-2.09 because LaTeX 2$_\varepsilon$'s version will not work with LaTeX-2.09. The main job is done by `\adl@hcline` after the target columns are `\span`-ed 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 `\adl@org@cline`, original version of `\cline`, after closing the `\noalign` opened by `\cdashline`.

```
\def\cdashline#1{\noalign{\ifnum0=‘}\fi
        \@ifnextchar[%]
                    {\adl@cdline[#1]}%
                    {\adl@cdline[#1][\dashlinedash/\dashlinegap]}}
\def\adl@cdline[#1-#2]{\global\adl@cla#1\relax
        \global\advance\adl@cla\m@ne
        \ifnum\adl@cla>\z@ \global\let\@gtempa\adl@cdlinea
        \else              \global\let\@gtempa\adl@cdlineb\fi
        \global\adl@clb#2\relax
        \global\advance\adl@clb-\adl@cla \ifnum0=‘{\fi}
        \@gtempa{-\arrayrulewidth}}
\def\adl@cdlinea{\multispan\adl@cla &\multispan\adl@clb \unskip \adl@hcline}
\def\adl@cdlineb{\multispan\adl@clb \unskip \adl@hcline}
```

```
\def\adl@inactivecdl[#1-#2][#3]{\ifnum0=‘{\fi}\adl@org@cline{#1-#2}}
```

\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-ed columns by \adl@draw. As we will discussed 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).

```
\def\adl@hcline#1[#2/#3]{\@tempdima#2\relax \@tempdimb#3\relax
        \adl@draw\adl@vrule\hskip\hbox \cr
        \noalign{\global\adl@finaldepth\z@ \ifdim#1=\z@\else \vskip#1\fi}}
```

\firsthdashline
\lasthdashline

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
\adl@idefflhdl
\adl@firsthdashline
\adl@lasthdashline

The macros \adl@first/lasthdashline, however, are defined in a tricky manner to replace \hline in \first/lasthline with;

\adl@hdashline\adl@ihdashline[⟨*dash*⟩/⟨*gap*⟩]

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 \exapndafter and the parts preceding and following \hline are extracted. Then the preceding part ⟨*p*⟩, the calling sequence of \adl@hdashline, and the following part ⟨*f*⟩ are connected to be the body of \adl@first/lasthdashline. Thus we define \adl@firsthdashline as follows.

```
\def\adl@firsthdashline[#1/#2]{%
    ⟨p⟩
    \adl@hdashline\adl@ihdashline[#1/#2]
    ⟨f⟩}
```

```
\ifadl@usingarypkg
\def\firsthdashline{\adl@hdashline{\ifnum0=‘{\fi}\adl@firsthdashline}}
\def\lasthdashline{\adl@hdashline{\ifnum0=‘{\fi}\adl@lasthdashline}}

\def\adl@defflhdl#1{\def\@tempa{#1}
        \expandafter\adl@idefflhdl}
\def\adl@idefflhdl#1\hline#2\@nil{%
        \@namedef\@tempa[##1/##2]{#1\adl@hdashline\adl@ihdashline[##1/##2]#2}}
\adl@defflhdl{adl@firsthdashline}\firsthline\@nil
\adl@defflhdl{adl@lasthdashline}\lasthline\@nil
\fi

%%^L
```

4.10 End of Environment

\endarray
\endtabular
\endtabular*

The macros to close the array/tabular environment, \endarray and \endtabular(*), are modified so that they invoke \adl@endarray to draw vertical lines just before closing \halign, and \adl@arrayrestore to restore registers and data structures \global-ly modified in the environment.

```
%% End of Environment

\def\endarray{\adl@endarray \egroup \adl@arrayrestore \egroup}
\def\endtabular{\adl@endarray \egroup \adl@arrayrestore \egroup $\egroup}
\expandafter\let\csname endtabular*\endcsname\endtabular

```

\adl@endarray
\adl@rows
\adl@addvl
\adl@vlrowL
\adl@vlrowR
\adl@vlrow

The macro \adl@endarray at first closes the last row by \crcr. If this \crcr has real effect, we have to invoke \adl@@cr to perfrom our own end-of-row operations. We assume that the \crcr is effective if either \adl@height or \adl@depth has a non-zero value[9].

Then the rows to draw vertical lines $L_1, \ldots, L_n$;

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

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

\adl@vl{β_k}{$\tau_k - \beta_k$}{δ_k}{ξ_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@makevlrL and \adl@makevlrR shares common macros, they conceptually have the following interface.

\adl@vlrow = \adl@makevlrL/R(\adl@rows : $\langle R^L$ *or* $R^R \rangle$,
\adl@currentcolumn : $\langle$*start column*$\rangle$,
\adl@addvl : $\langle$*macro to add an element*$\rangle$)

Thus they are invoked as;

\adl@vlrowL = \adl@makevrL(\adl@rowsL, 1, \adl@addvlL)
\adl@vlrowR = \adl@makevrR(\adl@rowsR, \adl@columns, \adl@addvlR)

Finally, after constructed rows for vertical lines are put by \adl@drawvl, a vertical skip of −\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.

```
\def\adl@endarray{\crcr \noalign{
        \ifdim\adl@height=\z@
        \ifdim\adl@depth=\z@  \else \adl@@cr\z@ \fi
                             \else \adl@@cr\z@ \fi
        \let\adl@vl\relax
        \def\adl@vlrow{}\adl@currentcolumn\@ne
                \let\adl@rows\adl@rowsL
                \let\adl@addvl\adl@addvlL
                \adl@makevlrL \global\let\adl@vlrowL\adl@vlrow
        \def\adl@vlrow{}\adl@currentcolumn\adl@columns
                \let\adl@rows\adl@rowsR
                \let\adl@addvl\adl@addvlR
                \adl@makevlrR \global\let\adl@vlrowR\adl@vlrow
```

[9] The author confesses that this rule is not strict and the introduction of a switch could improve the strictness.

```
        \global\let\adl@vl\adl@@vl}%
        \adl@drawvl
        \noalign{\vskip-\adl@finaldepth}%
        \omit\vrule\@width\z@\@height\z@\@depth\adl@finaldepth\cr}

```

`\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`.

```
\def\adl@arrayrestore{%
        \global\adl@height\adl@heightsave
        \global\adl@depth\adl@depthsave
        \global\adl@currentcolumn\adl@currentcolumnsave
        \global\adl@totalheight\adl@totalheightsave
        \global\let\adl@rowsL\adl@rowsLsave
        \global\let\adl@rowsR\adl@rowsRsave
        \global\let\adl@colsL\adl@colsLsave
        \global\let\adl@colsR\adl@colsRsave}

%%^L
```

4.11 Drawing Vertical Lines

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

R^L : `\adl@rowsL` R : `\adl@rows` R' : `\@tempb` Λ : `\adl@vlrowL`
Γ : `\adl@columns` γ : `\adl@currentcolumn`
τ : `\@tempcnta` β : `\@tempcntb` δ : `\adl@dash` ξ : `\adl@gap`
H : `\adl@totalheight`
conn : `\ifadl@connected` *double* : `\ifadl@doublerule`

`\adl@makevlrL` The macro `\adl@makevlrL` corresponds to the line (2) and (30)–(36). Its right-edge counterpart `\adl@makevlrR` has the same correspondance but the lines (1)–(2) are;

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

and (30)–(35) are;

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

```

%% Drawing Vertical Lines

\def\adl@makevlrL{\adl@makevlr
        \ifadl@doublerule
```

```
(1)  Λ ← ⟨⟩; R ← R^L; γ ← 1;
(2)  while γ ≤ Γ do begin
(3)      τ ← H; β ← H; δ ← −1; ξ ← −1;
(4)      conn ← false; double ← false;; R′ ← ⟨⟩
(5)      while R ≠ ⟨⟩ do begin
(6)          ⟨r, R⟩ ← R;
(7)          ⟨C, h⟩ ← r;
(8)          if C = ⟨⟩ then
(9)              add(τ, β, δ, ξ);
(10)         elseif C ≠ ⟨connect⟩ begin
(11)             ⟨e, C′⟩ = C; ⟨c, d, g⟩ = e;
(12)             if c = γ then begin
(13)                 if d = δ ∧ g = ξ then begin
(14)                     if ¬conn then begin
(15)                         τ ← β; conn ← true;
(16)                     end;
(17)                 end;
(18)                 else begin
(19)                     add(τ, β, δ, ξ);
(20)                     δ ← d; ξ ← g; τ ← β; conn ← true;
(21)                 end;
(22)                 if C′ = ⟨⟨γ, ?, ?⟩,?⟩ then   double ← true;
(23)                 C ← C′;
(24)             end;
(25)             else   add(τ, β, δ, ξ);
(26)         end;
(27)         β ← β − h; R′ ← ⟨R′, ⟨C, h⟩⟩
(28)     end;
(29)     add(τ, β, δ, ξ); R ← R′;
(30)     if double then   Λ ← ⟨Λ, \hskip\doublerulesep⟩;
(31)     else begin
(32)         γ ← γ + 1;
(33)         if γ > Γ then   Λ ← ⟨Λ, \hfil⟩;
(34)         else            Λ ← ⟨Λ, \hfil&\omit⟩;
(35)     end;
(36) end;
(37)
(38) procedure add(τ, β, δ, ξ) begin
(39)     if conn then begin
(40)         Λ ← ⟨Λ, ⟨β, τ − β, δ, ξ⟩⟩; conn ← false;
(41)     end;
(42) end;
```

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

```
                \edef\adl@vlrow{\adl@vlrow \hskip\doublerulesep}%
                \let\next\adl@makevlrL
        \else
                \advance\adl@currentcolumn\@ne
                \ifnum\adl@currentcolumn>\adl@columns \let\next\relax
                        \edef\adl@vlrow{\adl@vlrow \hss}%
                \else \let\next\adl@makevlrL
                        \edef\adl@vlrow{\adl@vlrow \hss &\omit}%
        \fi\fi\next}
\def\adl@makevlrR{\adl@makevlr
        \ifadl@doublerule
                \edef\adl@vlrow{\hskip\doublerulesep \adl@vlrow}%
                \let\next\adl@makevlrR
        \else
                \advance\adl@currentcolumn\m@ne
                \ifnum\adl@currentcolumn=\z@ \let\next\relax
                        \edef\adl@vlrow{\hss \adl@vlrow}%
                \else \let\next\adl@makevlrR
                        \edef\adl@vlrow{&\omit \hss \adl@vlrow}%
        \fi\fi\next}

```

`\adl@makevlr` The macro `\adl@makevlr` corresponds to the lines (3)–(4) and (29).

```
\def\adl@makevlr{\@tempcnta\adl@totalheight \@tempcntb\adl@totalheight
        \adl@dash\m@ne \adl@gap\m@ne
        \adl@connectedfalse \adl@doublerulefalse \def\@tempb{}%
        \expandafter\adl@imakevlr\adl@rows\@nil;%
        \adl@addvl
        \edef\adl@rows{\@tempb}}

```

`\adl@imakevlr` `\adl@iimakevlr` `\adl@endmakevlr` The macro `\adl@imakevlr`⟨*r*⟩`;` corresponds to the lines (5)–(6), and the macro `\adl@iimakevlr`(⟨*C*⟩`/`⟨*h*⟩`)` to (7) and (27).

```
\def\adl@imakevlr#1;{\def\@tempa{#1}\ifx\@tempa\@nnil \let\next\relax
        \else \adl@iimakevlr#1\let\next\adl@imakevlr \fi \next}
\def\adl@iimakevlr(#1/#2){\let\@elt\adl@iiimakevlr
        \let\adl@connect\adl@@connect
        \let\adl@endmakevlr\adl@endmakevlrcut
         #1\adl@endmakevlr
        \let\@elt\relax \let\adl@connect\relax
        \advance\@tempcntb-#2\edef\@tempb{\@tempb(\@tempc/#2);}}

```

`\adl@iiimakevlr` `\adl@ivmakevlr` `\adl@vmakevlr` `\adl@endmakevlrcut` `\adl@endmakevlrconn` `\adl@@connect` The correspondance of the lines (8)–(29) is a little bit complicated. As shown above, `\adl@iimakevlr` expands C attaching the sentinel `\adl@endmakevlr`.

1. If $C \neq \langle\rangle$ and $C \neq \langle \mathit{connect} \rangle$, C has at least one `\@elt`⟨*c*⟩⟨*d*⟩⟨*g*⟩ which is made `\let`-equal to `\adl@iiimakevlr` by `\adl@iimakevlr`. Thus the lines (10)–(21) and (25) are performed by `\adl@iiimakevlr`.

   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 $C = \langle \mathit{connect} \rangle$, i.e. it has only one element `\adl@connect`, the macro `\adl@@connect` is invoked because it is `\let`-equal to `\adl@connect`. The macro do nothing but implict $C \leftarrow C$ ($= \langle \mathit{connect} \rangle$) and eating the sentinel.

3. If $C = \langle\rangle$, `\adl@endmakevlrcut` that is `\let`-equal to the sentinel `\adl@endmakevlr` is invoked to perform (8)–(9) and implicit $C \leftarrow C$ ($= \langle\rangle$).

```
\def\adl@iiimakevlr#1#2#3{\let\@elt\adl@ivmakevlr \let\next\relax
        \ifnum#1=\adl@currentcolumn\relax
                \let\adl@endmakevlr\adl@endmakevlrconn
                \@tempswafalse
                \ifnum#2=\adl@dash\relax
                \ifnum#3=\adl@gap\relax
                        \@tempswatrue
                \fi\fi
                \if@tempswa
                        \ifadl@connected\else
                                \@tempcnta\@tempcntb \adl@connectedtrue \fi
                \else
                        \adl@addvl
                        \adl@dash#2\relax \adl@gap#3\relax
                        \@tempcnta\@tempcntb \adl@connectedtrue
                \fi
        \else
                \adl@addvl
                \def\next{\adl@vmakevlr\@elt{#1}{#2}{#3}}%
        \fi\next}
\def\adl@ivmakevlr#1#2#3{%
        \ifnum#1=\adl@currentcolumn \adl@doubleruletrue \fi
        \adl@vmakevlr\@elt{#1}{#2}{#3}}
\def\adl@vmakevlr#1\adl@endmakevlr{\def\@tempc{#1}}
\def\adl@endmakevlrcut{\adl@addvl \def\@tempc{}}
\def\adl@endmakevlrconn{\def\@tempc{}}
\def\adl@@connect\adl@endmakevlr{\def\@tempc{\adl@connect}}

```

`\adl@addvlL` `\adl@addvlR` The macro `\adl@addvlL` corresonds to the lines (38)–(42), i.e. the procedure *add*. The macro `\adl@addvlR` performs simlar operations, but its conceptual code is the following.

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

```
\def\adl@addvlL{\ifadl@connected
        \advance\@tempcnta-\@tempcntb
```

```
        \edef\adl@vlrow{\adl@vlrow
                \adl@vl{\number\@tempcntb}{\number\@tempcnta}%
                        {\number\adl@dash}{\number\adl@gap}}%
        \adl@connectedfalse \fi}
\def\adl@addvlR{\ifadl@connected
        \advance\@tempcnta-\@tempcntb
        \edef\adl@vlrow{\adl@vl{\number\@tempcntb}{\number\@tempcnta}%
                                {\number\adl@dash}{\number\adl@gap}\adl@vlrow}%
        \adl@connectedfalse \fi}

```

`\adl@drawvl` `\adl@@vl` `\adl@vl@leftskip` `\adl@vl@rightskip`

After the the macros `\adl@vlrowL` and `\adl@vlrowR` are constructed, they are expanded to draw vertical lines by `\adl@drawvl`. Prior to the expansion, the macro `\adl@drawvl` globally defines `\adl@vl@leftskip` and `\adl@vl@rightskip`, which are the amount of negative spaces inserted to the left/right of a vertical line, as follows.

$$\texttt{\adl@vl@leftskip} = \begin{cases} \texttt{\arrayrulewidth}/2 & \text{if } \texttt{\ifadl@zwrule} \\ 0 & \text{else if leftside} \\ \texttt{\arrayrulewidth} & \text{otherwise} \end{cases}$$

$$\texttt{\adl@vl@rightskip} = \begin{cases} \texttt{\arrayrulewidth}/2 & \text{if } \texttt{\ifadl@zwrule} \\ 0 & \text{else if rightside} \\ \texttt{\arrayrulewidth} & \text{otherwise} \end{cases}$$

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[10].

Then the macros `\adl@vlrowL` and `\adl@vlrowR` are expanded. These macros will have `\adl@vl`, which is made `\let`-equal to `\adl@@vl` prior to the expansion, to draw a vertical line. The macro `\adl@@vl`$\langle\beta\rangle\langle\lambda\rangle\langle\gamma\rangle\langle\delta\rangle$ draws a sloid line if $\gamma = 0$ or a dash-line otherwise in a `\vbox` of $\lambda = \tau - \beta$ high and `\raise`-s it by β. The method to draw a dash line in the `\vbox` is analogous to that for holizontal line shown in §4.9, except that a line is surrounded by horizontal spaces of `\adl@vl@leftskip` and `\adl@vl@rightskip`.

```
\def\adl@drawvl{%
        \omit \relax \ifadl@zwvrule
                        \gdef\adl@vl@leftskip{.5\arrayrulewidth}%
                        \global\let\adl@vl@rightskip\adl@vl@leftskip
                \else   \global\let\adl@vl@leftskip\z@
                        \global\let\adl@vl@rightskip\arrayrulewidth
                \fi \adl@vlrowL \cr
        \omit \relax \ifadl@zwvrule
                        \gdef\adl@vl@leftskip{.5\arrayrulewidth}%
                        \global\let\adl@vl@rightskip\adl@vl@leftskip
                \else   \global\let\adl@vl@leftskip\arrayrulewidth
                        \global\let\adl@vl@rightskip\z@
                \fi \adl@vlrowR \cr}

\def\adl@@vl#1#2#3#4{\vbox to\z@{\vss\hbox{%
        \hskip-\adl@vl@leftskip
```

[10] Before v1.54, the horizontal spaces was not inserted if `\ADLsomewide` and thus disconnected lines were not aligned vertically.

```
        \raise#1sp\vbox to#2sp{
                \ifnum#3=\z@
                        \hrule height#2sp depth\z@ width\arrayrulewidth
                \else   \@tempdima#3sp \@tempdimb#4sp
                        \adl@draw\adl@hrule\vskip\vbox
                \fi}%
        \hskip-\adl@vl@rightskip}}}

%%^L
```

4.12 Drawing Dash-lines

`\adl@vrule` `\adl@hrule`

As explained later, horizontal and vertical lines are drawn by a common macro `\adl@draw` 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.

```
%% Draw Dash Lines (\adl@vrule/\adl@hrule, \hskip/\vskip, \hbox/\vbox)

\def\adl@vrule#1{\vrule\@width#1\@tempdima\@height\arrayrulewidth\relax}
\def\adl@hrule#1{\hrule\@height#1\@tempdima\@width\arrayrulewidth\relax}
```

`\adl@drawi` `\adl@drawii` `\adl@drawiii` `\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.

$\langle rule \rangle${1/2} $\langle skip \rangle(g/2)$
`\xlearders`$\langle box \rangle${$\langle skip \rangle(g/2)$ $\langle rule \rangle${1} $\langle skip \rangle(g/2)$}
$\langle skip \rangle$(0 `plus 1fil minus 1fil`)
$\langle skip \rangle(g/2)$ $\langle rule \rangle${1/2}

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

$\langle rule \rangle${1/2} $\langle skip \rangle(g/2)$
$\langle box \rangle${$\langle skip \rangle(g/2)$ $\langle rule \rangle${1} $\langle skip \rangle(g/2)$} $\langle skip \rangle(-d-g)$
`\xlearders`$\langle box \rangle${$\langle skip \rangle(g/2)$ $\langle rule \rangle${1} $\langle skip \rangle(g/2)$}
$\langle skip \rangle$(0 `plus 1fil minus 1fil`)
$\langle skip \rangle(-d-g)$ $\langle box \rangle${$\langle skip \rangle(g/2)$ $\langle rule \rangle${1} $\langle skip \rangle(g/2)$}
$\langle skip \rangle(g/2)$ $\langle rule \rangle${1/2}

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`.

```
\def\adl@drawi#1#2#3{%
        #1{.5}#2.5\@tempdimb
        \xleaders#3{#2.5\@tempdimb #1{1}#2.5\@tempdimb}%
                #2\z@ plus1fil minus1fil\relax
        #2.5\@tempdimb #1{.5}}
\def\adl@drawii#1#2#3{%
        \setbox\adl@box#3{#2.5\@tempdimb #1{1}#2.5\@tempdimb}%
        #1{.5}#2.5\@tempdimb
        \copy\adl@box #2-\@tempdima #2-\@tempdimb
        \xleaders\copy\adl@box#2\z@ plus1fil minus1fil\relax
        #2-\@tempdima #2-\@tempdimb \copy\adl@box
        #2.5\@tempdimb #1{.5}}
\def\adl@drawiii#1#2#3{{\let\xleaders\cleaders \adl@drawi#1#2#3}}
\let\adl@draw\adl@drawi

```

`\ADLdrawingmode` The macro `\ADLdrawingmode{`$\langle m \rangle$`}` 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.

```
\def\ADLdrawingmode#1{\ifcase #1%
        \let\adl@draw\adl@drawi \or
        \let\adl@draw\adl@drawi \or
        \let\adl@draw\adl@drawii \or
        \let\adl@draw\adl@drawiii \else
        \let\adl@draw\adl@drawi \fi}

%%^L
```

4.13 Shorthand Activation

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

```

%% Shorthand Activation

\def\adl@Array{\adl@inactivefalse \array}
\def\adl@Tabular{\adl@inactivefalse \tabular}
\def\adl@Tabularstar{\adl@inactivefalse \@nameuse{tabular*}}
\def\adl@Longtable{\adl@inactivefalse \longtable}

```

`\@notdefinable` `\adl@notdefinable` Before making `\Array` etc. `\let`-equal to `\adl@Array` etc., we have to check if these macros 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.

```
\begingroup
\def\@notdefinable{\gdef\adl@notdefinable{}}
```

```
\@ifdefinable\Array\relax
\@ifdefinable\Tabular\relax
\expandafter\@ifdefinable\csname Tabular*\endcsname\relax
\ifx\longtable\undefined\else \@ifdefinable\Longtable\relax \fi
\endgroup
```

`\Array` `\Tabular` `\Tabular*` `\Longtable` `\endArray` `\endTabular` `\endTabular*` `\endLongtable`

If `\adl@notdefinable` is `\undefined` indicating that all `\Array` etc. are definable, we `\let` them be equal to `\adl@Array` etc. We also `\let` ending macros `\endArray` etc. be equal to `\endarray` etc. Note that `\Longtable` and `\endLongtable` are defined only when longtable is loaded, and `\endLongtable` is `\def`-ined as (not being `\let`-equal to) `\endlongtable` because its definition of our own is not given yet.

Otherwise, we complain with a warning message put by `\PackageWarning` if it is defined (i.e. LaTeX 2_ε) or `\@warning` otherwise (i.e. LaTeX-2.09).

```
\ifx\adl@notdefinable\undefined
        \let\Array\adl@Array
        \let\Tabular\adl@Tabular
        \expandafter\let\csname Tabular*\endcsname\adl@Tabularstar
        \let\endArray\endarray
        \let\endTabular\endtabular
        \expandafter\let\csname endTabular*\endcsname\endtabular
        \ifx\longtable\undefined\else
                \let\Longtable\adl@Longtable
                \def\endLongtable{\endlongtable}
        \fi
\else
\begingroup
\ifx\longtable\undefined
\def\@tempa{Array and Tabular are not defined because one of them\MessageBreak
        has been defined}
\else
\def\@tempa{Array/Tabular/Longtable are not defined because \MessageBreak
        one of them has been defined}
\fi
\ifx\PackageWarning\undefined
        \def\MessageBreak{^^J}
        \@warning\@tempa
\else
        \let\on@line\empty
        \PackageWarning{arydshln}\@tempa
\fi
\endgroup
\fi
```

`\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`.

```
\def\ADLnoshorthanded{%
        \let\Array\relax
```

```
        \let\Tabular\relax
        \expandafter\let\csname Tabular*\endcsname\relax
        \let\endArray\relax
        \let\endTabular\relax
        \expandafter\let\csname endTabular*\endcsname\relax
        \ifx\longtable\undefined\else
                \let\Longtable\relax
                \let\endLongtable\relax \fi}

```

\adl@act@arrayclassz
\adl@act@tabclassz
\adl@act@classz
\adl@act@@startpbox
\adl@act@@endpbox
\adl@act@endpbox
\adl@act@cr
\adl@act@argcr
\adl@act@cline
\adl@act@endarray
\adl@act@hline
\adl@act@ihdashline
\adl@act@cdline
\adl@act@@vlineL
\adl@act@@vlineR

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[11].

```
\let\adl@act@arrayclassz\@arrayclassz
\let\adl@act@tabclassz\@tabclassz
\ifadl@usingarypkg \let\adl@act@classz\@classz \fi
\let\adl@act@@startpbox\@@startpbox
\let\adl@act@@endpbox\@@endpbox
\let\adl@act@endpbox\@endpbox
\let\adl@act@cr\adl@cr
\let\adl@act@argcr\adl@argcr
\let\adl@act@cline\cline
\let\adl@act@endarray\adl@endarray
\let\adl@act@hline\adl@hline
\let\adl@act@ihdashline\adl@ihdashline
\let\adl@act@cdline\adl@cdline
\let\adl@act@@vlineL\adl@@vlineL
\let\adl@act@@vlineR\adl@@vlineR

%%^L
```

4.14 Compatibility with colortab

\adl@CC@
\CC@

The package colortab has a macro;

\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` stuffs by `\adl@arraysave` before the height measurement and restore them by `\adl@arrayrestore` after that.

```

%% Compatibility with colortab

```

[11] Alternatively, we may define `\adl@act@arrayclassz` in place of `\@arrayclassz` but the author chose this way to minimize the possiblity of *enbug*.

```
\def\adl@CC@#1#2#3{%
  \ifcolortab
    \noalign{%
      \adl@arraysave
      \setbox\CT@box=\vbox{#1#3\crcr\egroup}%
      \adl@arrayrestore
      \CT@dim=\ht\CT@box
      \global\advance\CT@dim by \dp\CT@box
      \def\CT@next{}%
      \futurelet\next\CT@columncolor#2&\@nil}%
    \CT@next\cr
    \noalign{\vskip-\CT@dim}%
  \fi
  #3}
\ifx\ColortabLoaded\undefined\else
\let\CC@\adl@CC@
\fi

%%^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 chunks 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, we skip everything if longtable is not in use, or we have undefined-error when we refer the definitions in it.

```

%% Compatibility with longtable: initialization

\ifx\longtable\undefined \let\next\endinput \else\let\next\relax \fi
\next

```

Next, the following switch and `\dimen` register are declared.

`\ifadl@LTfirstpage`

- `\ifadl@LTfirstpage` is tested in `\output` routine to examine if the page being put has the first fraction of a `longtable`.

`\adl@LTpagetotal`

- `\adl@LTpagetotal` is set to `\pagetotal` just before the first portion of a `longtable` is added to the main vertical list. Since the `\box255` has items preceding the `\longtable` and its first fraction, we can obtain the height of the first fraction by subtracting `\adl@LTpagetotal` from the height plus depth of `\box255`.

```
\newif\ifadl@LTfirstpage
\newdimen\adl@LTpagetotal
```

\adl@LT@array
\LT@array
\adl@discard

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@idashline and \adl@discard are made \let-equal to its longtable version \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 \hdasnline sequence.

```
\let\adl@LT@array\LT@array
\def\LT@array{\adl@arrayinit \adl@LTfirstpagetrue
        \let\adl@discard\relax \let\adl@ihdashline\adl@LTihdashline
        \ifadl@inactive \adl@LTinactivate \fi
        \adl@LT@array}
\def\adl@LTinactivate{\adl@inactivate
        \let\endlongtable\adl@org@endlongtable
        \let\LT@make@row\adl@org@LT@make@row
        \let\LT@echunk\adl@org@LT@echunk
        \let\LT@end@hd@ft\adl@org@LT@end@hd@ft
        \let\LT@kill\adl@org@LT@kill
        \let\LT@output\adl@org@LT@output
        \let\adl@ihdashline\adl@LTinactivehdl}
```

\adl@org@LT@make@row
\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 stuffs 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. Finally, the original version \adl@org@LT@make@row is called.

Table 2: Active and Inactive longtable Operations

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

```
\let\adl@org@LT@make@row\LT@make@row
\def\LT@make@row{\let\LT@make@row\adl@org@LT@make@row
        \adl@columns\adl@ncol
        \ifadl@usingarypkg\else
                \let\@@startpbox\adl@LTstartpbox
                \let\@startpbox\adl@LTstartpbox \fi
        \let\@@endpbox\adl@LTendpbox
        \let\@endpbox\adl@LTendpbox
        \adl@org@LT@make@row}

%%^L
```

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

4.15.2 Ending Chunks

\adl@org@endlongtable
\endlongtable
\adl@org@LT@echunk
\LT@echunk
\adl@LTlastrow

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

```

%% Compatibility with longtable: end chunk

\let\adl@org@endlongtable\endlongtable
\def\endlongtable{\adl@LTlastrow \adl@org@endlongtable}

```

```
\let\adl@org@LT@echunk\LT@echunk
\def\LT@echunk{\adl@LTlastrow \adl@org@LT@echunk}

\def\adl@LTlastrow{\crcr \noalign{
        \ifdim\adl@height=\z@
        \ifdim\adl@depth=\z@  \else \adl@@cr\z@ \fi
                             \else \adl@@cr\z@ \fi}}

```

\adl@org@LT@end@hd@ft
\LT@end@hd@ft
\adl@LThfsave
\adl@LTth
\\adl@LTth\LT@firsthead
\\adl@LTth\LT@head
\\adl@LTth\LT@lastfoot
\\adl@LTth\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

Another chunk ending macro is `\LT@end@hd@ft`⟨*box*⟩ to close a header/footer called by `\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 $R^{L/R}$ = `\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@LTth`⟨*box*⟩
`\\adl@rowsL`⟨*box*⟩
`\\adl@rowsR`⟨*box*⟩

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

`\\adl@LTth\LT@firsthead`

and so on. The saving operation is done by the macro `\adl@LThfsave`⟨*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.

```
\let\adl@org@LT@end@hd@ft\LT@end@hd@ft
\def\LT@end@hd@ft#1{\adl@LTlastrow
        \noalign{\edef\adl@LTth{\number\adl@totalheight}%
                \adl@LThfsave#1\adl@LTth \global\adl@totalheight\z@
                \adl@LThfsave#1\adl@rowsL\gdef\adl@rowsL{}%
                \adl@LThfsave#1\adl@rowsR\gdef\adl@rowsR{}}
        \adl@org@LT@end@hd@ft#1}
\def\adl@LThfsave#1#2{\expandafter\global\expandafter\let
        \csname\string#2\string#1\endcsname#2}

```

\adl@org@LT@kill
\LT@kill
\adl@LTkill
\adl@LTkillend

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 `\\` preceding `\kill`, 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` find 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.

```
\let\adl@org@LT@kill\LT@kill
\def\LT@kill{\adl@LTlastrow \noalign{
            \def\@tempb{}\expandafter\adl@LTkill\adl@rowsL\@nil\adl@rowsL
            \def\@tempb{}\expandafter\adl@LTkill\adl@rowsR\@nil\adl@rowsR
            \global\advance\adl@totalheight-\@tempcnta}%
        \adl@org@LT@kill}
\def\adl@LTkill#1;#2{\def\@tempa{#2}%
        \ifx\@tempa\@nnil\def\next{\adl@LTkillend#1}%
        \else\edef\@tempb{\@tempb#1;}\def\next{\adl@LTkill#2}\fi
        \next}
\def\adl@LTkillend(#1/#2)#3{\global\let#3\@tempb \@tempcnta#2\relax}

%%^L
```

4.15.3 Horizontal Lines and p-Boxes

`\LT@hline` `\adl@LTihdashline` `\adl@LTinactivehdl` `\adl@LThdlrow`

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`[12]. 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`. 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.

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*⟩`]`.

```

%% Compatibility with longtable: horizontal lines and p-boxes

\def\LT@hline{\noalign{\ifnum0=`}\fi
        \gdef\adl@LThdlrow{\multispan{\LT@cols}\unskip
                \leaders\hrule\@height\arrayrulewidth\hfill\cr}%
        \adl@LThdline}
\def\adl@LTihdashline[#1/#2]{%
        \gdef\adl@LThdlrow{\multispan{\LT@cols}\unskip
                \adl@hcline\z@[#1/#2]}%
        \adl@LThdline}
\def\adl@LTinactivehdl[#1/#2]{%
        \gdef\adl@LThdlrow{\multispan{\LT@cols}\unskip
                \leaders\hrule\@height\arrayrulewidth\hfill\cr}%
        \adl@LThdline}

```

`\adl@LThdline` `\adl@LTxhline` `\adl@LTixhline`

The macro `\adl@LThdline` called by above three macros first add the pseudo row information *connect*(`\arrayrulewidth`) to $R^{L/R}$ by `\adl@hline`[13] and insert a vertical penalty 10000 to inhibit page break between the horizontal line and preceding row. Then 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`

[12] In the original longtable, a sequence of three `\hline`-s are not recognized. This buggy feature is fixed in this implementation.

[13] Or do noting if inactive and thus it is `\let`-equal to `\@gobbletwo`.

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. 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 $R^{L/R}$.

```
\def\adl@LThdline{\adl@hline\adl@connect\arrayrulewidth \penalty\@M
        \ifnum0=‘{\fi}%
        \adl@LThdlrow
        \noalign{\ifnum0=‘}\fi
        \futurelet\@tempa\adl@LTxhline}
\def\adl@LTxhline{\ifx\@tempa\hline \adl@LTixhline
        \else\ifx\@tempa\hdashline \adl@LTixhline
        \else \penalty-\@lowpenalty \vskip-\arrayrulewidth
                \adl@hline\adl@discard{-\arrayrulewidth}%
                \adl@hline\adl@connect\arrayrulewidth
        \fi\fi \ifnum0=‘{\fi}%
        \adl@LThdlrow \noalign{\penalty\@M}}
\def\adl@LTixhline{\penalty-\@medpenalty \vskip\doublerulesep
        \adl@hline\relax\doublerulesep \global\let\adl@LThdlrow\@empty}
```

\adl@LTstartpbox
\adl@LTendpbox

Macros for opening/closing p-boxes are fairly simple. The macros \adl@LTstartpbox{⟨*w*⟩} and \adl@LTendpbox are \let-assigned to \@@startpbox and \@@endpbox by \LT@make@row. The former opens a p-box of w wide by our own \adl@act@@startpbox and performs a footnote related operation introduced by longtable. The latter closes the p-box by our own \adl@act@@endpbox and also performs the footnote stuffs. 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.

```
\def\adl@LTstartpbox#1{%
        \adl@act@@startpbox{#1}\let\@footnotetext\LT@p@ftntext}
\def\adl@LTendpbox{\adl@act@@endpbox \the\LT@p@ftn \global\LT@p@ftn{}}

%%^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 followings.

- After it inserts a vertical skip \LTpre, \endgraf is performed so that the skip contributes to \pagetotal[14].
- 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[15], and restored from \@tempcnta.
- 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.

[14] This modification is necessary for the original longtable, or it underestimates the room of the first page and leaves head and foot only.

[15] This is also necessary for the original version.

- The box `\LT@firsthead` is put by `\copy` rather than `\box` because it is referred in the `\output` routine.

This macro does not have inactive counterpart because the modification shown above is desirable (first two) or not-harmful[16] (last two) to the original version.

```
%% Compatibility with longtable: first chunk

\def\LT@start{%
        \let\LT@start\endgraf
        \endgraf \penalty\z@ \vskip\LTpre \endgraf
        \dimen@\pagetotal
        \advance\dimen@ \ht\ifvoid\LT@firsthead\LT@head\else\LT@firsthead\fi
        \advance\dimen@ \dp\ifvoid\LT@firsthead\LT@head\else\LT@firsthead\fi
        \advance\dimen@ \ht\LT@foot
        \dimen@ii\vfuzz \@tempcnta\vbadness
        \vfuzz\maxdimen \vbadness\@M
        \setbox\tw@\copy\z@
        \setbox\tw@\vsplit\tw@ to \ht\@arstrutbox
        \setbox\tw@\vbox{\unvbox\tw@}%
        \vfuzz\dimen@ii \vbadness\@tempcnta
        \advance\dimen@\ht
                \ifdim\ht\@arstrutbox>\ht\tw@\@arstrutbox\else\tw@\fi
        \advance\dimen@\dp
                \ifdim\dp\@arstrutbox>\dp\tw@\@arstrutbox\else\tw@\fi
        \advance\dimen@ -\pagegoal
        \ifdim \dimen@>\z@\vfil\break \fi
        \global\adl@LTpagetotal\pagetotal
        \global\@colroom\@colht
        \ifvoid\LT@foot\else
                \advance\vsize-\ht\LT@foot
                \global\advance\@colroom-\ht\LT@foot
                \dimen@\pagegoal\advance\dimen@-\ht\LT@foot\pagegoal\dimen@
                \maxdepth\z@
        \fi
        \copy\ifvoid\LT@firsthead \LT@head \else \LT@firsthead \fi
        \output{\LT@output}}

%%^L
```

4.15.5 Output Routine

`\adl@org@LT@output` `\LT@output`

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

[16] Logically, at least.

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 in \adl@LTdraw, they are put by \copy rather than \box.

```
%% Compatibility with longtable: output routine

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

\adl@LTdraw
\adl@LTinit
\adl@LTheadL
\adl@LTheadR
\adl@LTfootL
\adl@LTfootR

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

$$T = \begin{cases} \texttt{\adl@LTpagetotal} & \text{if } \texttt{\ifadl@LTfirstpage} \\ 0 & \text{otherwise} \end{cases}$$

$$t = \begin{cases} \texttt{\topskip} \text{ glue} & \text{if longtable is the first item of the page} \\ & (\neg(\texttt{\ifadl@firstpage} \wedge T{>}0)) \\ 0 & \text{otherwise} \end{cases}$$

$$H = H_{255} - t - T$$
$$H_b = H - H_h - H_t$$

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 1 sp high as its first item. Then `\box0` is `\vsplit` to 1 sp 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{`$\langle hf \rangle$`}`$\langle box \rangle$, where $\langle hf \rangle$ is `head` or `foot` and $\langle box \rangle$ is one of `\LT@firsthead`, `\LT@head` and $\langle foot \rangle$ (`\LT@lastfoot` or `\LT@foot`). This macro also copies the contents of weirdly named structure such as `\\adl@rowsL\LT@head` into `\adl@LTheadL` and so on[17] if $\langle box \rangle$ is not void. Otherwise, `\adl@LTheadL` etc. is kept to their initial value, `\@empty`.

Next, we make rows for vertical lines by `\adl@makevlrL/R` after extracting the leading part of $R^{L/R}$ corresponding to the *body* by the macro `\adl@LTsplit`$\langle R^{L/R} \rangle \langle R_h^{L/R} \rangle \langle R_f^{L/R} \rangle$, 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@makevlL/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@vlrowL` and `\adl@vlrowR` as in the ordinary case without `longtable`.

Then the rows are put into `\box0` by calling `\LT@bchunk` with `\adl@drawvl` (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 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 $\langle tail \rangle$ (`\vss` or nothing) to its end.

```
\def\adl@LTdraw#1#2{%
        \@tempswatrue
        \ifadl@LTfirstpage\ifdim\adl@LTpagetotal>\z@\@tempswafalse \fi\fi
        \if@tempswa
                \setbox\z@\vbox{\hrule height1sp\unvcopy\@cclv}
                \splittopskip\z@
                \setbox\@ne\vsplit\z@ to1sp\relax
                \@tempdima\ht\z@
        \else   \@tempdima\ht\@cclv \fi
        \advance\@tempdima\dp\@cclv
        \adl@totalheight\@tempdima
        \let\adl@LTheadL\@empty \let\adl@LTheadR\@empty
        \let\adl@LTfootL\@empty \let\adl@LTfootR\@empty
        \ifadl@LTfirstpage
                \global\adl@LTfirstpagefalse
                \advance\@tempdima-\adl@LTpagetotal
                \adl@totalheight\@tempdima
                \ifvoid\LT@firsthead
                        \adl@LTinit{head}\LT@head
                \else   \adl@LTinit{head}\LT@firsthead
```

[17] Copying by `\edef` can be replaced by `\let` with many `\expandafter` but it is not comprehensible.

```
                \fi
        \else   \adl@LTinit{head}\LT@head \fi
        \ifvoid#1%
                \adl@LTinit{foot}\LT@foot
        \else   \adl@LTinit{foot}#1\fi
        \let\adl@vl\relax \def\adl@discard{\adl@connect}%
        \def\adl@vlrow{}\adl@currentcolumn\@ne
                \adl@LTsplit\adl@rowsL\adl@LTheadL\adl@LTfootL
                \let\adl@addvl\adl@addvlL
                \adl@makevlrL \let\adl@vlrowL\adl@vlrow
        \def\adl@vlrow{}\adl@currentcolumn\adl@columns
                \adl@LTsplit\adl@rowsR\adl@LTheadR\adl@LTfootR
                \let\adl@addvl\adl@addvlR
                \adl@makevlrR \let\adl@vlrowR\adl@vlrow
        \let\adl@vl\adl@@vl
        \@tempcnta\LT@rows
        \LT@bchunk \adl@drawvl
        \LT@save@row\cr \egroup \setbox\@ne\lastbox \unskip \egroup
        \global\advance\c@LT@chunks\m@ne
        \global\LT@rows\@tempcnta
        \@tempdima\dp\@cclv
        \setbox\@cclv\vbox{\unvbox\@cclv \box\z@ \vskip-\@tempdima
                \hrule\@width\z@\@height\z@\@depth\@tempdima#2}}
\def\adl@LTinit#1#2{\ifvoid#2\else
        \advance\@tempdima-\csname\string\adl@LTth\string#2\endcsname sp%
        \expandafter\edef\csname adl@LT#1L\endcsname{%
                \csname\string\adl@rowsL\string#2\endcsname}%
        \expandafter\edef\csname adl@LT#1R\endcsname{%
                \csname\string\adl@rowsR\string#2\endcsname}\fi}

```

\adl@LTsplit
\adl@LTxsplit
\adl@LTrowrelax
\adl@LTrowdiscard
\adl@LTysplit
\adl@LTisplit
\adl@LTiisplit
\adl@LTsplitend

The macro `\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' (`\adl@rows`) until total heights of the elements summed in h (`\@tempdimb`) reaches to H_b (`\@tempdima`)[18] by a straight forward loop with the macros `\adl@LTisplit` to fetch the i-th element and `\adl@LTiisplit` to get h_i. Before moving, however, we have to remove discardable item(s)[19] 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 $R^{L/R}$.

```
\def\adl@LTsplit#1#2#3{\def\adl@rows{}\@tempdimb\z@
        \expandafter\adl@LTxsplit#1\@nil;%
        \edef\adl@rows{#2\adl@rows#3}%
        \global\let#1\@tempb}
```

[18] 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.

[19] Must be only one but the implementation allows two or more.

```
\def\adl@LTxsplit#1;{\def\@tempa{#1}%
        \ifx\@tempa\@nnil \def\@tempb{}\let\next\relax
        \else\ifx\adl@LTheadL\@empty \def\next{\adl@LTysplit#1}%
        \else \def\next{\adl@LTisplit#1;}\fi \fi
        \next}
\def\adl@LTrowrelax{\relax}
\def\adl@LTrowdiscard{\adl@discard}
\def\adl@LTysplit(#1/#2){\def\@tempa{#1}%
        \ifx\@tempa\adl@LTrowrelax \let\next\adl@LTxsplit
        \else\ifx\@tempa\adl@LTrowdiscard \let\next\adl@LTxsplit
        \else \def\next{\adl@LTisplit(#1/#2);}\fi \fi
        \next}
\def\adl@LTisplit#1;{\def\@tempa{#1}%
        \ifx\@tempa\@nnil \def\@tempb{}\let\next\relax
        \else\ifdim\@tempdimb<\@tempdima
                \adl@LTiisplit#1\let\next\adl@LTisplit
        \else   \def\next{\adl@LTsplitend#1;}\fi \fi
        \next}
\def\adl@LTiisplit(#1/#2){\edef\adl@rows{\adl@rows(#1/#2);}%
        \advance\@tempdimb#2sp}
\def\adl@LTsplitend#1;\@nil;{\def\@tempb{#1;}}
```

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 TEX 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`; and Craig Leech who reported the incompatibility problem, which was also reported by Uwe Jehmlich, Torge Thielemann and Florian Weig, and have waited for two years and a half (!) for the solution.

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 2_ε (1997/12/01) to which Johannes Braams and other authors also contributed. The author also refers array package (v2.3m) written by Frank Mittelbach and David Carlisle; colortab package (v0.9) written by Timothy van Zandt; and longtable package (v4.10) written by David Carlisle; to make the style compatible with those packages.