REPORT LAYOUT MANAGER SPECIFICATION

Author: Qiangsheng Wang

1. Introduction

The layout manager handles the block area(s) that it creates. The layout managers are set up from the formatting structure tree. The layout manager is essentially a bridge between the formatting objects and the area tree.

This spec defines the features the layout manager will provide.

The requirement includes:

- **Inline and block features** Currently designer only support every weak inline and block features. It's quite not easy to add more support with current structure for all logics are mixed with GEF together. With separate layout managers, we can support more inline and block features.
- **Table layout algorithm** The table layout algorithm is quite complex, please refer to Tables section in design detail to get more inform about detail of its algorithm. With the table layout manager we can has support better CSS style table layout.
- **Consistent structure and behavior with CSS definition** with layout manager, we can follow CSS definition layout algorithm to get more consistent behavior.

2. Design Details

2.1 CSS processing model

As CSS defined, a user agent processes a source by going through the following steps:

1. Parse the source document and create a document tree.

2. Identify the target media type

3. Retrieve all style sheets associated with the document that are specified for the target media type.

4. Annotate every element of the document tree by assigning a single value to every property that is applicable to the target media type. Proper ties are assigned values according to the mechanisms described in the section on cascading and inheritance.

Part of the calculation of values depends on the formatting algorithm appropriate for the target media type. For example, if the target medium is the screen, user agents apply the visual formatting model.

5. From the annotated document tree, generate a formatting structure. Note that the CSS user agent does not alter the document tree during this phase. In particular, content generated due to style sheets is not fed back to the document language processor.

6. Transfer the formatting structure to the target medium (e.g., prints the results, display them on the screen, render them as speech, etc.)

The layout manager only concern features that listed in the step 5 and 6. The features include generate a formatting structure model tree, implements the box model, implements the media independent layout algorithm, and etc.

For those page dependent features like paging, etc, will not include in this design.

2.2 Layout processing

The role of the layout managers is to create the area by using the information from the model tree. The layout managers decide the position, weight and width of the element in the area. They also manage the handling of breaks and spacing between areas.

Normally any object that creates a block area will have an associated layout manager. Other cases are tables and lists, these objects will also have layout managers that will manager the group of layout managers that make up the object.

The layout process is handled by a set of layout managers. The layout traverses the model tree in pre-order, depth-first ordering.

2.3 Render

A renderer is primarily designed to convert a given model tree into the output format. Each renderer is given an area model, which is created by layout manger, to render to its output format. The area model is simply a representation of the pages and the placement of text and graphical objects on those pages.

2.4 Using Layout Managers

A layout manager is essentially a bridge between the model objects and the render. When render try to convert the model tree into the output format, the layout manager is used to determine position, width, height and other information.

The calling on layout manager can happens when organizing the layout on a page or editing the layout by user.

2.5 GEF based render and layout

The report designer is based on eclipse GEF framework and need a GEF based render to show the report structure on UI.

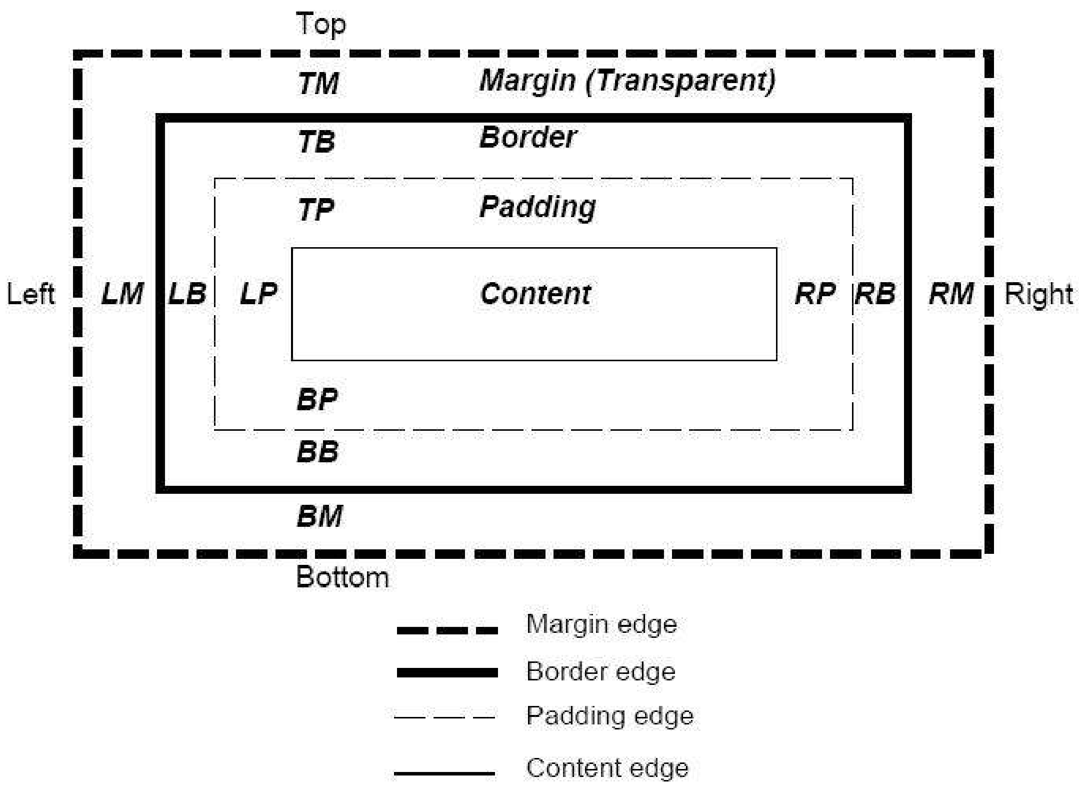
The GEF IFigure has the role of render for report designer UI. The layout managers will provider layout information for IFigure(the render) to let it place and draw report element on the screen by using GEF API.

3. Features

3.1 Box model

The CSS box model describes the rectangular boxes that are generated for elements in the document tree and laid out according to the visual formatting model.

Each box has a *content area* (e.g., text, an image, etc.) and optional surrounding *padding*, *border*, and *margin* areas; the size of each area is specified by properties defined below. The following diagram shows how these areas relate and the terminology used to refer to pieces of margin, border, and padding:



3.2 Visual formatting model

The visual formatting model describe: how user agents process the document tree for visual media (paged or not paged median can all be visual media).

In the visual formatting model, each element in the document tree generates zero or more boxes according to the box model. The layout of these boxes is governed by:

- Box dimensions and type
- Positioning scheme (normal flow, float, and absolute positioning).
- Relationships between elements in the document tree.
- External information (e.g., view port size, intrinsic dimensions of images, etc.).
- Normal flow, floats, and absolute positioning
- Calculating widths, height, and margins for inline and block.

3.3 Table

The CSS table model is based on the HTML 4.0 table model, in which the structure of a table closely parallels the visual layout of the table. In this model, a table consists of an optional caption and any number of rows of cells. The table model is said to be "row primary" since authors specify rows, not columns, explicitly in the document language. Columns are derived once all the rows have been specified the first cell of each row belongs to the first column, the second to the second column, etc.). Rows and columns may be grouped structurally and this grouping reflected in presentation (e.g., a border may be drawn around a group of rows). Thus, the table model consists of tables, captions, rows, row groups, columns, column groups, and cells.

Visual layout of table contents

Internal table elements generate rectangular boxes with content and borders. Cells have padding as well. Internal table elements do not have margins. The visual layout of these boxes is governed by a rectangular, irregular grid of rows and columns.

Table width algorithms

Fixed table layout

With this (fast) algorithm, the horizontal contents of the cells; it only depends and borders or cell spacing.

Automatic table layout

In this algorithm (which generally requires no more than two passes), the table's width is given by the width of its columns. This algorithm reflects the behavior of several popular HTML user agents at the writing of this specification. UAs are not required to implement this algorithm to determine the table layout in the case that 'table-layout' is 'auto'; they can use any other algorithm. This algorithm may be inefficient since it requires the user agent to have access to all the content in the table before determining the final layout and may demand more than one pass.

Table height algorithms

The height of a table is given by the 'height' property for the 'table' or 'inline-table' element. A value of 'auto' means that the height is the sum of the row heights plus any cell spacing or borders. Any other value specifies the height explicitly; the table may thus be taller or shorter than the height of its rows. CSS 2.1 does not specify rendering when the specified table height differs from the content height, in particular whether content height should override specified height; if it doesn't, how extra space should be distributed among rows that add up to less than the specified table height; or, if the content height exceeds the specified table height, whether the UA should provide a scrolling mechanism.

3.4 Borders

The separated borders model

The table in the figure below could be the result of a style sheet like this:

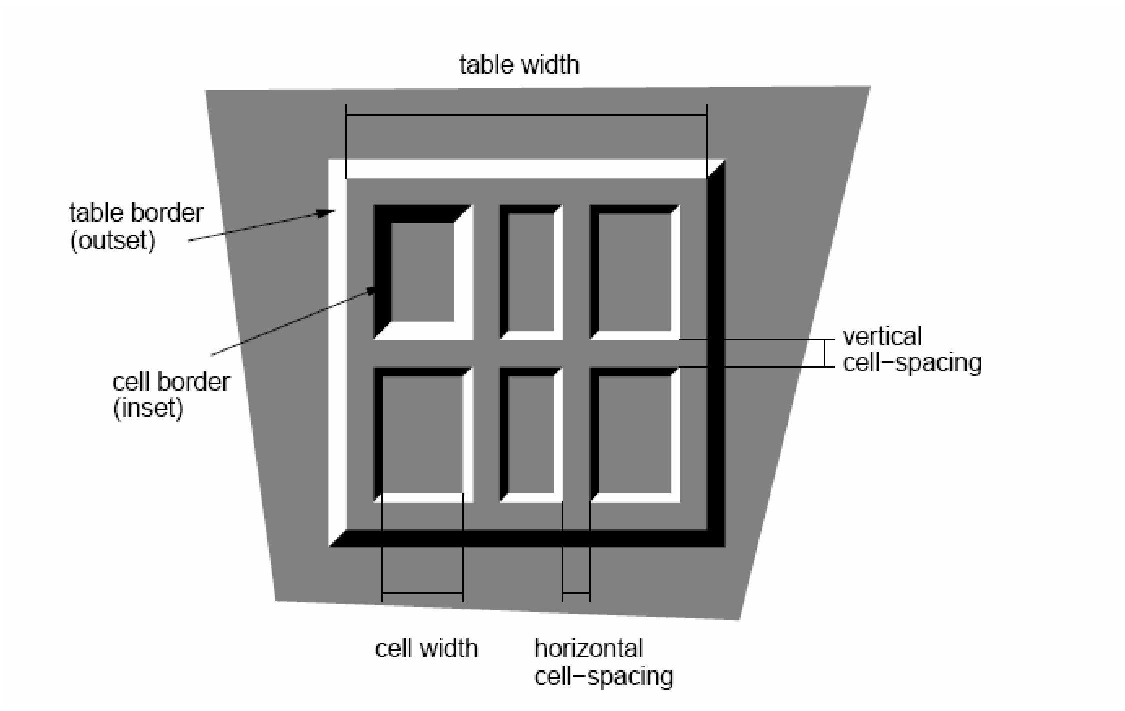
Table {border: outset 10pt;

Border-collapse: separate;

Border-spacing: 15pt}

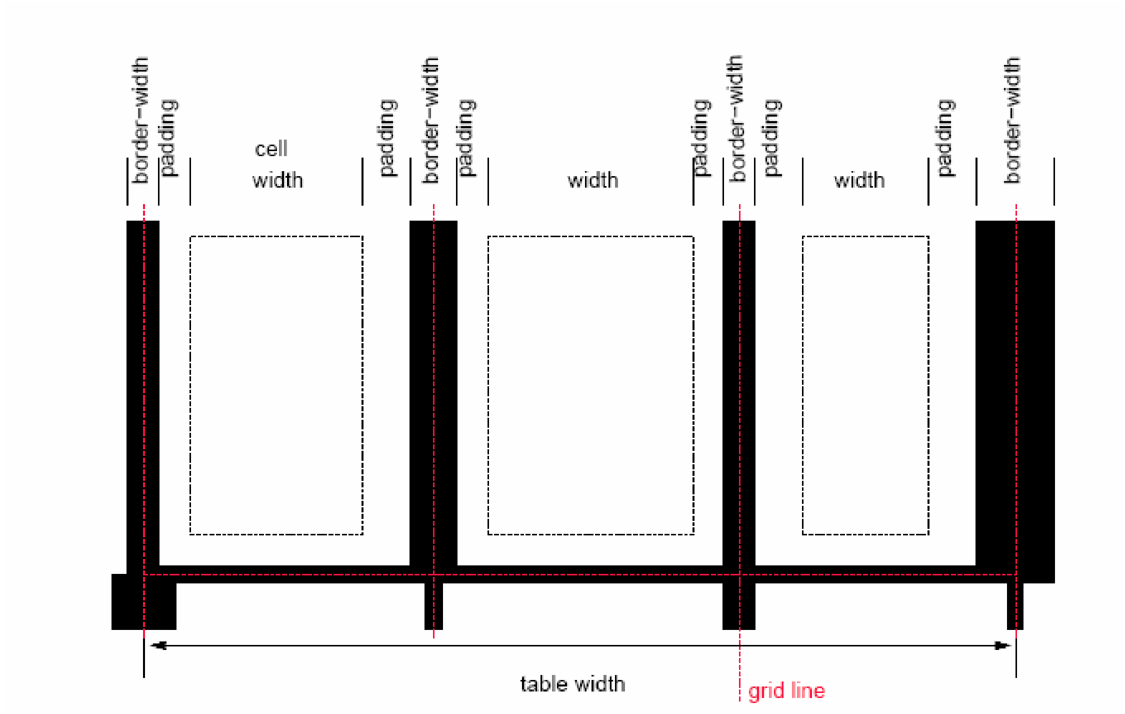
Td {border: inset 5pt}

Td.special {border: inset 10pt} /* The top-left cell */



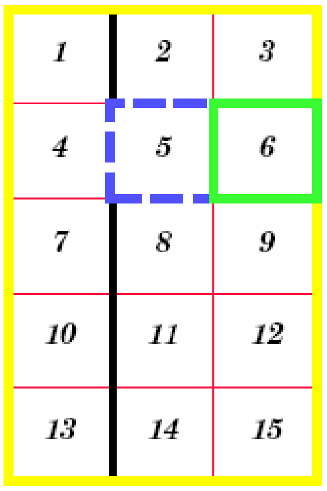
The collapsing border model

In the collapsing border model, it is possible to specify borders that surround all or part of a cell, row, row group, column, and column group. Borders for HTML's "rule" attribute can be specified this way.



Border conflict resolution

In the collapsing border model, borders at every edge of every cell may be specified by border properties on a variety of elements that meet at that edge (cells, rows, row groups, columns, column groups, and the table itself), and these borders may vary in width, style, and color. The rule of thumb is that at each edge the most "eye catching" border style is chosen, except that any occurrence of the style 'hidden' unconditionally turns the border off.



An example of a table with collapsed borders.

4. APIS

4.1 Calling on layout manager

Calling on layout manager can happens when organizing the layout on a page or editing the layout by user.

Following is the pseudo code to start layout traverse on a page creation or a page size changing.

```
createPage(Page currentPage)
{
    //layout the page
currentPage().layout
// draw the page with layout information
drawMargin();
drawBorder();
…..
}

updateHeight()
{ // change causes container re-layout
   if (affectContainingBlock)
```

```
    {
      containingBlock().updateHeight();
      return;
    }
    // re-layout itself
    layout();
}
```

4.2 Layout traverse

Following is the pseudo code to layout a table and its contents. This is a typical usage case to show how layout is working

```
public void layout()
{
    // calculate min/max width for table columns and cells
     FOR_EACH_CELL(r, c, cell)
     {
        cell->calcMinMaxWidth();
     }
    calcColWidth();
    setCellWidths();

    // layout rows
    FOR_EACH_ROW
    {
      FOR_EACH_CELL_IN_THE_ROW
      {
            // layout children
            // cell will traverse its childre to let every children be layouted.
            cell.layout();
      }
    }
```

4.3 ILayoutManger interface

```
    public interface ILayoutManager
    {
    /**
    * This function calculates the minimum & maximum width that the object
    * can be set to.
    *
    */
        public  void calcMinMaxWidth() { }

      /**
       * Calculates the actual width of the object (only for non inline
       * objects)
       */
        public void calcWidth() {}

     /**
       * This function should cause the Element to calculate its
      * width and height and the layout of it's content
      *
      * If element has child item, the Element should also layout all it's
      * direct child items.
      */
        public void layout();

       /**
        * this function get's called, if a child changed it's geometry
        * (because an image got loaded or some changes in the DOM...)
        */
        virtual void updateSize();
}
```

Some important layout managers can be

- ReportFlowLayout
- TableLayout
- ListLayout
- BodyLayout

4.4 GEF based render and IFigure

The IFigure interface of GEF can be used to implement GEF based render. The render will draw the report element with the help of layout manager.

The render also handles the color, font, background, border, padding, margin, size and other elements properties by using Draw2d API of eclipse.

4.5 BoxFigure

The BoxFigure is the abstract class to implement CSS box model. The BoxFigure provides functions to support padding, border, and margin areas.

All other figure need padding, border and margin features will inherit this class.

5. References

Cascading Style Sheets, level 2 revision 1 (CSS 2.1) specification

Mozilla project

Monqueror: AN HTML BROWSER RUNING ON MINIGUI