Access to Layout Information with JavaScript

PDFreactor allows JavaScript access to layout information via the proprietary object ro.layout. This layout information can mainly be accessed in the form of descriptions of elements inside the document.

1. Descriptions

1.1. BoxDescription

The Box description describes the position and dimensions of the rectangles of a box as well as information about the page the element is on and the text it contains. When getting the box description of an element, the "getBoxDecriptions(element)" method will return an array of box descriptions because an element that is split over several pages, regions or columns will have several box descriptions.

```
let element = document.querySelector("#myElem");
let boxDescriptions = ro.layout.getBoxDescriptions(element);
if (boxDescriptions.length > 0) {
    let boxDescription = boxDescriptions[0];
}
```

There are 4 different rectangles that can be accessed from the box description. These are the margin rectangle, padding rectangle, border rectangle and the content rectangle. These rectangles contain the position and dimensions of these specific areas of a box. The unit of these dimensions can be specified when getting the rectangles. The point of origin of these rectangles is the upper left corner of the page content rectangle but you can also get these rectangles with the point of origin being the upper left corner of the whole page.

```
let element = document.querySelector("#myElem");
let boxDescriptions = ro.layout.getBoxDescriptions(element);
if (boxDescriptions.length > 0) {
    let boxDescription = boxDescriptions[0];
    let marginRect = boxDescription.getMarginRect("cm");
    let paddingRect = boxDescription.getPaddingRect("in");
    let borderRect = boxDescription.getBorderRect("pt");
    let contentRect = boxDescription.getContentRect("px");

    let marginRectInPage = boxDescription.getMarginRectInPage("cm");
    let paddingRectInPage = boxDescription.getPaddingRectInPage("in");
    let borderRectInPage = boxDescription.getBorderRectInPage("pt");
    let contentRectInPage = boxDescription.getContentRectInPage("px");
}
```

1.2. Page Description

The page description describes the dimensions of a page and its rectangles. It can be accessed either by the page index or from a box description of an element:

```
let pageDesc = ro.layout.getPageDescription(0);

let element = document.querySelector("#myElem");
let boxDescriptions = ro.layout.getBoxDescriptions(element);
if (boxDescriptions.length > 0) {
    let boxDescription = boxDescriptions[0];
    let pageDesc = boxDescription.pageDescription;
}
```

The same kind of rectangles that can be accessed from the box description also can be accessed from the page description. These rectangles contain information about the dimensions of the page.

```
let pageDescription = ro.layout.getPageDescription(0);
let marginRect = pageDescription.getMarginRect("cm");
let paddingRect = pageDescription.getPaddingRect("in");
let borderRect = pageDescription.getBorderRect("pt");
let contentRect = pageDescription.getContentRect("px");
```

There are also two kinds of ranges that can be accessed from the page description. The "range" is a DOM range where the start- and end container are the most deeply nested nodes at the respective page breaks, while the "rangeShallow" is a DOM range where the start and end container are the least deeply nested nodes at the respective page breaks. For example if you had a p element inside a div element at a page break, the range would return the p element as the end container, while the "rangeShallow" would return the div element as the end container.

```
let pageDescription = ro.layout.getPageDescription(0);
let range = pageDescription.range;
let rangeShallow = pageDescription.rangeShallow;
```

1.3. Line Description

The line description contains information about a line of text. Because an element can contain more than one line of text the "lineDescriptions" property of the box description returns an array of line descriptions.

```
let element = document.querySelector("#myElem");
let boxDescriptions = ro.layout.getBoxDescriptions(element);
if (boxDescriptions.length > 0) {
    let boxDescription = boxDescriptions[0];
    let lineDescriptions = boxDescription.lineDescriptions;
}
```

The line description contains information about its content rectangle as well as its baseline position. The "getBaselinePosition" method returns the vertical distance between the baseline position of the line and the top of the content rectangle of the box containing the line.

```
let element = document.querySelector("#myElem");
let boxDescriptions = ro.layout.getBoxDescriptions(element);
if (boxDescriptions.length > 0) {
    let boxDescription = boxDescriptions[0];
    let lineDescriptions = boxDescription.lineDescriptions;
    if (lineDescriptions.length > 0) {
        let lineDescription = lineDescription[0];
        let contentRect = lineDescription.getContentRect("cm");
        let baseLinePos = lineDescription.getBaselinePosition("in");
    }
}
```

2. Use Cases

2.1. Page-/Region-/ColumnIndex

A frequent use case for the ro.layout object is to find out the page number of a specific element by accessing the page index from the box description of the element. However as the page index starts from zero, one has to be added to it to get the page number.

```
let element = document.querySelector("#myElem");
let boxDescriptions = ro.layout.getBoxDescriptions(element);
if (boxDescriptions.length > 0) {
    let boxDescription = boxDescriptions[0];
    let pageNumber = boxDescription.pageIndex + 1;
}
```

You can find out the index of the column/region an element is in by getting the column-/regionIndex from the box description. This index starts at 0 for the first column/region and is not reset on new pages or by column spans. Alternatively you can also get the column-/regionIndexLocal which is reset on new pages and by column spans.

```
let element = document.querySelector("#myElem");
let boxDescriptions = ro.layout.getBoxDescriptions(element);
if (boxDescriptions.length > 0) {
    let boxDescription = boxDescriptions[0];
    let columnIndex = boxDescription.columnIndex;
    let columnIndexLocal = boxDescription.columnIndexLocal;
}
```

2.2. Styling Elements on Left/Right Pages

Elements can be styled, depending on if they are on a right or a left page. This can be achieved by getting the pageLeft attribute of the box description of an element.

```
let element = document.querySelector("#myElem");
let boxDescriptions = ro.layout.getBoxDescriptions(element);
if (boxDescriptions.length > 0) {
    let boxDescription = boxDescriptions[0];
    if (boxDescription.pageLeft) {
        // Apply styles for elements on a left page.
    } else {
        // Apply styles for elements on a right page.
    }
}
```

2.3. Breaks Inside Elements

Because elements have several box descriptions if they are split over multiple pages, you can check if an element contains a break by checking the length of its box description array.

```
let element = document.querySelector("#myElem");
let boxDescriptions = ro.layout.getBoxDescriptions(element);
if (boxDescriptions.length > 0) {
    let containsBreak = boxDescriptions.length > 1;
}
```

2.4. Space Left after an element

Another frequent use case is finding out how much space is left after a specific element. This can be done by comparing the margin rectangle of the element to the content box of this page. For example a break could then be inserted if the remaining space is below a certain threshold.

```
let element = document.querySelector("#myElem");
let boxDescriptions = ro.layout.getBoxDescriptions(element);
if (boxDescriptions.length > 0) {
    let boxDescription = boxDescriptions[0];
    let pageDescription = boxDescription.pageDescription
    let boxMarginRect = boxDescription.getMarginRect("cm");
    let pageContentRect = pageDescription.getContentRect("cm");
    let spaceLeft = pageContentRect.bottom - boxMarginRect.bottom;
}
```

Please note that the content rectangle of the page is used here rather than the margin rectangle, because the margin rectangle of a page includes the page margin boxes.

2.5. Preventing Text Overflow

Another use for the ro.layout object is to find out if text is overflowing. This can be done by getting the last line description of a box description and comparing its bottom coordinate to the height of the box description. You then reduce the font-size of this element and repeat this process until the text is not overflowing or a specific font-size minimum is reached.

```
let element = document.querySelector("#myElem");
let boxDescriptions = ro.layout.getBoxDescriptions(element);

if (boxDescriptions.length > 0) {
    var height = boxDescriptions[0].getContentRect().height;
    var textOffset = getTextOffset(element);

    // Get the font size of the element.
    var style = window.getComputedStyle(element, null).getPropertyValue('font-size');
    var fontSize = parseFloat(style);
    console.log(fontSize);

    while (textOffset > height && fontSize >= 10) {
        fontSize -= 0.1;
        element.style.fontSize = fontSize + "px";
        textOffset = getTextOffset(element);
    }
}

function getTextOffset(paragraph) {
    var lines = ro.layout.getBoxDescriptions(paragraph)[0].lineDescriptions;
    var lastLine = lines[lines.length - 1];

    return lastLine.getContentRect().bottom;
}
```