Table of Contents | minLevel = 1 | maxLevel = 7

Introduction

We are now using a library from iTextPdf to produce PDF reports from Velocity templates. This library has more features, is easier to use, and does not have the performance issues we experienced with our current solution, AsposePDF.

The new implementation simplifies the templates in many ways. There's no need to use custom header/footer tags, and the new library interprets HTML style parameters correctly, so all the formatting, margin control, page decorations, etc can be specified in vanilla HTML and CSS styles.

This is a quick-start document explaining the basics of writing HTML templates for Velocity PDFs using the new library.

Basic HTML Templates

A basic template is plain HTML with a <body> section. Opening the HTML file in a browser is the fastest, easiest way to check the look and layout (keeping in mind that any embedded Velocity code will not render, and there will be no page breaks nor will headers/footers render if they are included). In order to see all the PDF features, the report must be uploaded and run in Connect. Remember that when these HTML templates are uploaded to Connect, they must have a .vm extension.

Here is a simple template for reporting on all the item names in the current project:

```
<html>
<head>
    <title>All Item Names</title>
</head>

<body>
    Project: $project.name
    <ul>
        #foreach( $itemId in
$documentSource.getActiveDocumentIdsInProject($project.id) )
           #set ($item = $documentSource.getDocument($itemId))
           <li>$item.name</li>
        #end
    </ul>
</body>
</html>
```

Styling

CSS styles are applied as usual. These styles should be honored in the PDF, but if there are problems with the way a particular style is rendered in the PDF, please let the Quick Wins team know so we can look into the issue.

This template is the same item name report above, but with style: adding margins, setting the background color, specifying the font, and prepending each item name with a green check mark.

```
<html>
<head>
```

```
<style>
  .item-list li {
      position: relative;
      list-style-type: none;
      padding-left: 2.5rem;
      margin-bottom: 0.5rem;
  }
  .item-list li:before {
     content: '';
     display: block;
     position: absolute;
     left: 0;
     top: -2px;
     width: 5px;
     height: 11px;
     border-width: 0 2px 2px 0;
     border-style: solid;
     border-color: #00a8a8;
     transform-origin: bottom left;
     transform: rotate(45deg);
   }
   html {
      -webkit-font-smoothing: antialiased;
      font-family: "Helvetica Neue", sans-serif;
      font-size: 62.5%;
   }
   body {
      width: 790px;
      font-size: 1.6rem; /* 18px */
      background-color: #efefef;
      color: #324047
   }
   html, body, section {
      height: 100%;
   }
   section {
      max-width: 400px;
      margin-left: 10px;
      margin-right: 20px;
      display: flex;
      align-items: center;
   }
   div {
      margin: auto;
   }
</style>
    <title>All Item Names</title>
</head>

<body>
<section>
    <div>
        <h2>Items for Project $project.name</h2>
        <ul class="item-list">
            #foreach( $itemId in
                $documentSource.getActiveDocumentIdsInProject($project.id)
            )
                #set ($item =
                    $documentSource.getDocument($itemId))
                <li>$item.name</li>
            #end
```

```
        </ul>
    </div>
</section>
</body>
</html>
```

Setting Paged Media Attributes

The CSS rule *@page* allows report writers to specify how pages are laid out. Details like page margins, page size and orientation, headers, footers, page numbering, and more can be defined within this rule.

As an example, adding these @page rules to the style section sets the page size to 8.5 by 11 inches, all margins to 2cm, adds a top margin of 10cm on the first page only, and ensures page breaks never occur immediately after an <h2> element.

```
<style>

    @page {
        size: 8.5in 11in;
        margin: 2cm;
    }

    @page :first {
        margin-top: 10cm;
    }

    h2 { page-break-after : avoid }

</style>
```

Reports can be generated with a landscape orientation by including the landscape keyword when specifying the page dimensions:

```
 <style>
        @page {
            size: A4 landscape;

            @bottom-right {
                font-family: Arial, Helvetica, sans-serif;
                font-size: 10.0pt;
                content: "Page " counter(page) " of " counter(pages);
            }
        }
</style>
```

For a full description of the page formatting available, see any of several online resources.

Adding Headers and/or Footers

The @page rule can be used to include custom headers and footers on every page, or just on selected pages. The process is:

1. Define an element in the html body section with a class name uniquely identifying it as a header or footer. This

element defines the look of the header/footer.
2. Include the CSS for the header/footer class in the style section. This stye definition must include the *position: running(...)* attribute.
3. In the @page rule block, include positioning information for the header/footer

Here is an implementation example for adding a simple header and footer:

1. Define header/footer HTML elements

In this simple example, these are just text elements that appear on the page. A more complex example is in the next section.

```
<div class='header'>An Awesome Report</div>
<div class='footer'>Copyright 2021, Jama Software</div>
```

The divs defining the header/footer should be at the top of the <body> section, appearing immediately after the <body> tag. The class names do not have to be "header" or "footer", but do need to be unique for the header/footer elements.

2. Include CSS for header/footers

```
div.header {
   display: block; text-align: center;
   position: running(header);
}

div.footer {
   font-family: Courier, sans-serif;
   font-size: 10.0pt;
   display: block; text-align: center;
   position: running(footer);
}
```

Styles for the header/footer html elements are defined here. The important elements in these styles are the *position* properties. They must be set to *running* with the class name for the header/footer specified. *running* indicates to the layout system that the class is removed from the normal page layout flow and is available to be referenced in the @page block.

3. Define the header/footer in the @page block

In the @page rule block described above, set the positioning for the header/footer and declare the content that will appear in this position. There are 16 possible position attributes, but the ones most likely to be used for headers/footers are: *@top-left, @top-center, @top-right, @bottom-left, @bottom-center,* and *@bottom-right.* The *content:element* attributes must reference the class name for the header/footer.

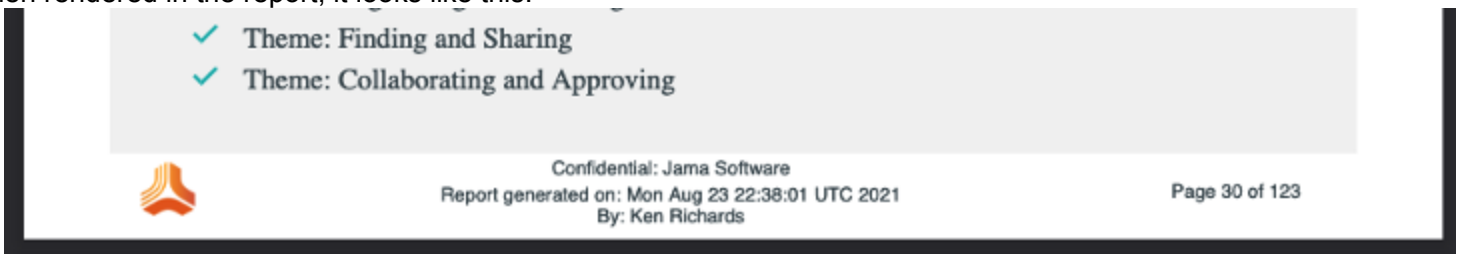
```
@page {
   size: A4;
   @top-center { content: element(header) }
   @bottom-center { content: element(footer) }
}
```

Below is a more complex footer that includes an image, dynamic content, and page info. It follows the same pattern, but includes a couple of special techniques.

This HTML describes a footer that includes an image, static text, dynamic data fetched from Velocity, and paging info:

```
<div class="footer">
    <table class="footer-table">
        <tr>
            <td rowspan="2">
                <img height="30" width="30"
src="https://www.jamasoftware.com/media/gravatar/jama-avatar.png"
                    alt="Jama Logo">
            </td>
            <td>
                Confidential: Jama Software
            </td>
            <td rowspan="2">
                Page <span id="pageNumber"></span> of <span id="totalPages"></span>
            </td>
        </tr>
        <tr>
            <td>Report generated on: $dateTool.getSystemDate()<br>By:
$userSource.currentUser.fullName</td>
        </tr>
    </table>
</div>
```

When rendered in the report, it looks like this:



The only unusual element here is the display of the page numbers. Two components are defined in the <style> section and used on line 12: *pageNumber* and *totalPages*.

The page numbering components are defined in the <style> section like this:

```
#pageNumber::after {
   content: counter(page)
}

#totalPages::after {
   content: counter(pages)
}
```

This code renders the default page counter after any HTML element with the id *"pageNumber"* and renders the total page count after any HTML element with the *"totalPages"* id. *counter(page)* and *counter(pages)* are page counters that are built into CSS.

Images

Images in Velocity PDFs come from two sources:

- Images that are part of an item and stored in Connect (e.g. Wiris equations or images uploaded into the rich text areas)

- External images

External images are included using the <img> tag. The src property must refer to a web server or file system that is accessible by Connect when the report is rendered.

The library we use for rendering PDFs does not fully support all SVGs. In order to ensure that all SVGs included in a report are rendered correctly, Connect automatically converts them to PNGs before they are include in the report. When the third-party library fully supports rendering for all SVGs, we will be able to render them in reports without conversion.

Tables

- Centered and appropriately aligned tables in PDFs are going to be reliant on css, so ensuring that the width of a table is defined as a percentage instead of calculated into pt or px. `width:80%` for example.
- Word breaks also need to be handled at the TableCell and TableHeader level, using `overflow-wrap`

Controlling Page Size and Margins

Page size is specified in the @page rule block as seen above using the *size:* element. There are several preset values available, but custom widths and heights may also be set.

Margins are controlled using CSS values as usual with HTML pages. For example this CSS will narrow the margins for a text-only section of the report:

```
.intro-text {
   margin-left: 100px;
   margin-right:200px;
}
```

Default Fonts vs Embedded Fonts

The iText pdfHTML add-on by default supports these 14 Standard Type 1 fonts:

- Times-Roman, Helvetica, Courier, Symbol, Times-Bold, Helvetica-Bold, Courier-Bold, ZapfDingbats, Times-Italic, Helvetica-Oblique, Courier-Oblique, Times-BoldItalic, Helvetica-BoldOblique, Courier-BoldOblique
- These fonts can be referenced in the HTML without needing a CSS styling definition or external link

Fonts other than the default Connect system fonts may be included by embedding them in the HTML template. Below is an example of embedding and using a font in the <style> section of the HTML template. As with external images, the URL for the font must point to a location that is locally accessible by Connect when the report is run.

```
 @font-face {
   font-family: 'rakkasregular';
   src: url('/Users/krichards/Desktop/Rakkas/Rakkas-Regular.ttf') format('ttf'),
   url('/Users/krichards/Desktop/Rakkas/Rakkas-Regular.ttf') format('ttf');
   font-weight: normal;
   font-style: normal;
}

.intro-text {
   margin-left: 100px;
   margin-right:200px;
   font-family: 'rakkasregular';
}
```

For fonts that are remotely accessible by Connect when the report is run, a <link> element must be used that points to the url for the remote font. This <link> element is used within the <head> element of the template, but is entirely separate from the <style> element. The <link> element should be formatted as follows:

```
<link rel="stylesheet"
      href="https://fonts.googleapis.com/css?family=Tangerine">
```

The family name can then be used as the value for any other font-family attributes in the template.
Note: The CSS @import rule is not compatible with iText pdfHTML

Compiled Example Template

All the techniques discussed in this document are in this complete Velocity template:

```
<html>
<head>
    <style>

        @page {
            size: A4;
            @top-center {
                content: element(header)
            }
            @bottom-center {
                content: element(footer)
            }
        }

        #pageNumber::after {
            content: counter(page)
        }

        #totalPages::after {
            content: counter(pages)
        }


        div.header {
            display: block;
            text-align: center;
            position: running(header);
        }

        div.footer {
            font-family: Arial, Helvetica, sans-serif;
            font-size: 8.0pt;
            display: block;
            text-align: center;
            position: running(footer);
        }

        .item-list li {
            position: relative;
            list-style-type: none;
```

```
    padding-left: 2.5rem;
    margin-bottom: 0.5rem;
}

.item-list li:before {
    content: '';
    display: block;
    position: absolute;
    left: 0;
    top: -2px;
    width: 5px;
    height: 11px;
    border-width: 0 2px 2px 0;
    border-style: solid;
    border-color: #00a8a8;
    transform-origin: bottom left;
    transform: rotate(45deg);
}

html {
    -webkit-font-smoothing: antialiased;
    font-family: "Helvetica Neue", sans-serif;
    font-size: 62.5%;
}

body {
    width: 790px;
    font-size: 1.6rem; /* 18px */
    background-color: #efefef;
    color: #324047
}

html,
body,
section {
    height: 100%;
}

section {
    max-width: 400px;
    margin-left: 10px;
    margin-right: 20px;
    display: flex;
    align-items: center;
}

div {
    margin: auto;
}

.footer-table  {
    width: 100%;
    height: 100px;
    vertical-align: middle;
    horiz-align: center;
    margin-bottom: 5px;
}

@font-face {
    font-family: 'rakkasregular';
    src: url('/Users/krichards/Desktop/Rakkas/Rakkas-Regular.ttf')
```

```
format('ttf'),
            url('/Users/krichards/Desktop/Rakkas/Rakkas-Regular.ttf') format('ttf');
            font-weight: normal;
            font-style: normal;
        }

        .report-info {
            margin-left: 150px;
            margin-right: 100px;
            font-family: 'rakkasregular';
            background-color: white;
            page-break-after: always;
        }

    </style>

    <title>All Item Names</title>
</head>

<body>
<div class="header">All Item Names in the Project</div>
<div class="footer">
    <table class="footer-table">
        <tr>
            <td rowspan="2">
                <img height="30" width="30"
src="https://www.jamasoftware.com/media/gravatar/jama-avatar.png"
                     alt="Jama Logo">
            </td>
            <td>
                Confidential: Jama Software
            </td>
            <td rowspan="2">
                Page <span id="pageNumber"></span> of <span id="totalPages"></span>
            </td>
        </tr>
        <tr>
            <td>Report generated on: $dateTool.getSystemDate()<br>By:
$userSource.currentUser.fullName</td>
        </tr>
    </table>
</div>
<h2>Summary of Report</h2>
<div class="report-info">
    <p>Text goes here</p>
</div>
<section>
    <div>
        <h2>Items for Project $project.name</h2>
        <ul class="item-list">
            #foreach( $itemId in
                $documentSource.getActiveDocumentIdsInProject($project.id)
            )
                #set ($item =
                    $documentSource.getDocument($itemId))
                <li>$item.name</li>
            #end
        </ul>
    </div>
</section>
</body>
```

```
</html>
```

Troubleshooting

Spacing Inconsistencies

When the iText pdfHTML add-on is generating the final PDF document, the spacing determined between lines separated by `<br />` tags within one <p> element vs lines separated by two distinct <p> elements will be unequal. The `<br />` tags result in less space than fully separate <p> elements. This should be kept in mind when writing the template itself, but it should also be kept in mind when thinking about the rich text fields in the Connect items themselves. These rich text fields are represented by underlying HTML that will get piped into the final HTML output from Velocity when running your template. If the rich text field contains lines of data separated by full -enter- breaks, they will appear with more space in the final PDF than lines of data that are separated by -shift+enter- breaks. This is because full -enter- breaks terminate and create new <p> elements whereas -shift+enter- breaks simply insert `<br />` tags without terminating the current <p> element.

Escaped Hexadecimal Failure

If you want to represent a character using its hexadecimal value equivalent (e.g. `\0000a0` for ` `), it is important to remember that any characters following the hexadecimal value can sometimes be mistakenly interpreted as part of the same value if there is no whitespace between them. However, one may be averse to separating them via a whitespace if a whitespace is not desired for the final resulting display. To assist with this problem, CSS allows for one single whitespace character following an escaped hexadecimal value to be used as a separator that it will ignore for the purposes of displaying. This normally wouldn't be required for back-to-back escaped hexadecimal values (e.g. `\0000a0\0000a0`), but it does seem to cause issues with iText by not treating the second as escaped. So, it is recommended that you use the free trailing whitespace character as a separator when writing escaped hexadecimal values to ensure that they are both escaped properly (`\0000a0 \0000a0`).