Ecma/TC39/2023/039

Guide to converting a Ecma standard from a ecmarkup generated HTML file into an Ecma format printable PDF

Allen Wirfs-Brock (2023 Editions)
allen@wirfs-brock.com

Introduction

This document describes the process used to create well-formatted printable PDF versions of Ecma TC39 HTML format standards documents created using ecmarkup. This process was first used for the 2022 editions of ECMA-262 and ECMA-402. This guide describes the processed used for the 2023 editions.

PDF renderings of TC39 specification are produced by augmenting the HTML files generated by ecmarkup with additional attributes and CSS rules defined by the W3C *CSS Paged Media Module Level 3*, *CSS Generated Content for Paged Media Module*, and *CSS Fragmentation Module Level 3* specifications. The augmented HTML is "printed" as a PDF document using a web browser and the Paged.js JavaScript polyfill library.

Browsers currently have only limited support for the CSS Paged Media features. The Paged.js pollyfill adds support for many of the missing features that are essential for formatting a book-length paginated document. As of 2023, Chromium-based browsers have the best Paged Media support and are the best hosts for Paged.js. For the 2023 editions of the TC39 standards, a Macintosh running Brave browser was used with Paged.js version 0.4.1 to product the PDFs. Current and legacy versions of the Paged.js pollyfill may be downloaded from: https://unpkg.com/pagedjs/dist/ .

Step-by-step Guide

Step 1 collect files

You need the ecmarkup generated html files and any locally referenced supporting files such as CSS and image files. Note that these files in either their original or modified forms are only needed during creation of the pdf. The Paged.js generated pdf will be self contained.

If the HTML file has already been published to the Ecma website you can browse to the Ecma-hosted document retrieve them. Alternatively, the files can be accessed from the https://tc39.es website anytime after the final draft has been approved by TC39; typically in early April. The files can be accessed via a URL schema of the form `https://tc39.es/<ecmaid>/<year>/`. For example, the 2023 editions can be accessed at https://tc39.es/ecma262/2023/ and https://tc39.es/ecma402/2023/.

Don't use a browser "Save page as..." operation to retrieve the files as it will convert internal `href`'s into into external URL's. Either use a command line tool (E.G. `wget` or `curl`) or the browser dev console to save the files.

If you don't already have them you will probably want to also retrieve (for comparison purposes) the https://tc39.es files for the previous years editions. The released PDF of the previous year is also useful to see the formatting decisions that were made.

It is also useful to have on hand the Paged Media augmented HTML files use to generate the previous years PDFs.

Step 2 modify the `<head>` of the html file for pdf generation

Update css links The `<head>` section of the html file may link to some css files that will not be needed for pdf generation. Other pdf specific css links probably need to be added. The following is an annotated example to how the links of the ecma-402-10 (2023) was updated. First here are links in the original html

(line breaks have been added for readability):

```html
<!doctype html>
<html lang="en-GB-oxendict">
<head><meta charset="utf-8">
<link rel="icon" href="img/favicon.ico">
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/highlight.js/11.0.1/styl
<link href="ecmarkup.css" rel="stylesheet">
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/highlight.js/8.4/styles/
```

and here is how they were updated:

```html
<!doctype html>
<html lang="en-GB-oxendict">
<head><meta charset="utf-8">
<link rel="icon" href="img/favicon.ico">
  <!--not used in 402-10: <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/lil
  <!--replaced by paged css variant: <link href="ecmarkup.css" rel="stylesheet"> -->
<link href="ecmarkup-ecmapdf-2023.css" rel="stylesheet">
  <!--not used in 402-10: <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/lil
```

The favicon is not needed for the PDF but there is no need to remove it.

Two css files related to highlight.js were linked, but ECMA-402-10 does not appear to use any styles they define (determined by searching the html file for `hljs-` which is the prefix of highlight.js defined styles; there were no hits). So, those links were removed. ECMA-262-14 does use highlight.js styles so for that document these links are kept.

Eliminating the links to highlight.js styles in documents that don't need them is not really necessary. In the future they probably should just be left in place.

The original ecmarkup generated html files have a link to `ecmarkup.css` which defines (most) styles used by the ecmarkup generated html. For printing, some of these styles need to be replaced or updated and some new print-related styles need to be added. This is accomplished by change the link to `ecmarkup.css` to instead link to `ecmarkup-ecmapdf-2023.css`. Note, that the replacement css also defines the PDF page layout template such as the placement of the Ecma page header image and the placement of page numbers and Ecma copyright on left and right side pages.

The replacement css file contains all the relevant css rules from `ecmarkup.css`. For the 2022 editions it excluded rules related to web-specific presentation and the specification browsing UI. For the 2023 edition, the replacement file includes both web screen and print rules with CSS @media queries used to distinguish/select rules that only apply to the web or print versions.

The year component of the file name indicates that it was derived from the `ecmarkup.css` as it existed when the TC39 standards for that year were released by Ecma. In subsequent years it will be necessary to check for any changes to `ecmarkup.css` and to propagate them into a revised `ecmarkup-ecmapdf-20XX.css`.

In both 2022 and 2023 the ecmarkup.css files used by ECMA-262 and ECMA-402 were slightly different. With one or both of the files having extra rules that did not appear in the other. Rather than creating two variants of `ecmarkup-ecmapdf-20XX.css` start by creating a common `ecmarkup.css` and use that to create a common `ecmarkup-ecmapdf-20XX.css`.

Add tags to load paged.js resources Paged.js is a polyfill that adds support for CSS Paged Media features to Chromium-based browsers.

```
 <!-- paged.js resources -->
   <!-- if you want the latest version -->
   <!-- <script src="https://unpkg.com/pagedjs/dist/paged.polyfill.js"></script> -->

  <!-- paged.js 0.4.1 was used for 2023 editions -->
  <script src="./pagedjs-4.1/paged.polyfill.js"></script>
  <link href="./pagedjs-4.1/interface.css" rel="stylesheet" type="text/css" />
```

For stability and reproducibility, use a local copy of paged.js. In future years, you will likely want to use the then current version.

Eliminated unneeded scripts Delete or comment out scripts that are only need to support interactive browsing of the HTML version:

```
<!-- not needed for pdf generation
<script src="ecmarkup.js"></script>
```

```
<script>
  if (location.hostname === 'tc39.es' && location.protocol !== 'https:') {
    location.protocol = 'https:';
  }
</script>
-->
```

If you retrieved the original HTML file from https://ecma-international.org it may contain scripts related to Google Analytics or other trackers. These can also be removed.

It probably isn't really necessary to remove such unneeded scripts but doing so eliminates any possibility of them interfering with the operations of paged.js.

Add some document specific parameters There are a few document specific parameters that are set via CSS variables and rules. Using these parameter eliminates the need to modify them in the body text. Placing this CSS early in the `<head>` makes it easy to find if it necessary to modify the parameters.

For ECMA-262 in 2023 they are:

```
<style>
  /* set some document specific parameters */

  :root { /*  set the publication year of this standard */
    --copyright-year: "2023";  /* <span class=year></span> */
    --ecma-edition: "14"; /* <span class=edition></span> */
  }
  /* if the PDF page number rendered for the page containing clause 1 is not 1, adjust the 
  @page clauses :first {
    /*counter-reset: page 1;*/
    counter-increment: page -9;
  }
</style>
```

For ECMA-402 in 2023 they are:

```
<style>
  /* set some document specific parameters */

  :root { /*  set the publication year of this standard */
    --copyright-year: "2023"; /* <span class=year></span> */
    --ecma-edition: "10"; /* <span class=edition></span> */
  }
  /* if the PDF page number rendered for the page containing clause 1 is not 1, adjust the 
  @page clauses :first {
    /*counter-reset: page 1;*/
    counter-increment: page -7;
  }
</style>
```

The values of these CSS variables can be inserted into the body of the document using the `<span>` tags as noted in the comments.

It would simplify things if the original authored content used these variables/spans and ecmarkup inserted their definitions. For example, for the PDF documents, we created a version of the ECMA copyright notices that uses the copyright year span.

In Ecma standards, pages are given decimal page numbers starting with page 1 for the the page containing the start of clause 1. Pages in the front-matter have Roman numeral page numbers. Paged.js only supports a single page counter that starts at the first physical page of the PDF. To get the decimal page numbers correct the page counter has to be reset when the first clause is reached. In theory setting `counter-reset: page 1;` should do that but it doesn't. However using `counter-increment` with the appropriate negative values does.

Eliminate unnecessary custom `<style>` rules Some ecmarkup generated HTML files (including those for ECMA-262 and ECMA-402) have a `<style>` element in their `<head>` that define styling rules or property definitions that are specific to that document or for some reason were not included in `ecmarkup.css`. These need to be reviewed to make sure they are consistent with the the needs of PDF print formatting. Preexisting rules/properties enabled by `@media print` or for pagination such as `page-break-before: always;`, or scrolling should be removed because different rules are defined in `ecmarkup-ecmapdf-20XX.css`, by subsequent conversion steps, or they are not applicable to printed documents.

For example, it is common to have rules such as these:

```
<style>
  /* custom styles */
  @media print {
    /* for the PDF */
    body.oldtoc {
      font-size: 80%;
    }
    .oldtoc var {
      color: #197124;
    }
  }
  #metadata-block {
    margin: 4em 0;
    padding: 10px;
    border: 1px solid #ee8421;
    page-break-after: always; /* so that the TOC, which appears next, is on a new page */
  }
  #sec-intro {
    page-break-before: always; /* so that the TOC, which appears previous, is on its own pag
  }
</style>
```

These rules all relate in some way to the old table of content style what was used for printing prior to switching to the use of Paged.js for PDF generation.

Most of these rules or properties could be safely deleted but it may be informative to keep them around. When rendering the document using Paged.js we never "display" elements tagged with the `oldtoc` CSS class so rules for that class are never applied.

The first two rules above used a `print` media query to ensure that they were only applied for printing. However, when they say `@media print` they don't mean the print formatting used by for the Paged.js generated version. We don't want Paged.js to use those rules. Paged.js provides a custom media type `pagedjs-ignore` that can be used for this purpose. It tells Paged.js to ignore the rules controlled by such a media query.

The last rule above is also specific to the old printed table of content, but that dependency is implicit. We can explicitly control it by moving it under the a `pagedjs-ignore` media query. Most of the properties in the `#meta-block` rule apply to both screen and Paged.js rendering of the document. But one of the properties is specific to the old toc formatting. That can be resulted by removing that property from the existing rule and placing it in a new `#meta-block` rule that is also under the `pagedjs-ignore` media query.

Applying these observation, we replace the above block of rules with the following:

```
<style>
  /* custom styles */
  @media print , pagedjs-ignore {
    /* for the old ugly PDF */
    body.oldtoc {
      font-size: 80%;
    }
    .oldtoc var {
      color: #197124;
    }
    #metadata-block {
        page-break-after: always; /* so that the old TOC, which appears next,
        is on a new page */
    }

    #sec-intro {
        page-break-before: always; /* so that the old TOC, which appears previous,
        is on its own page */
    }
  }

#metadata-block {
    margin: 4em 0;
    padding: 10px;
    border: 1px solid #ee8421;
    /*page-break-after: always;*/ /* so that the TOC, which appears next, is on a new page
}
</style>
```

Step 3 Reorganize the `<body>` of the HTML file for pdf generation

TC39 web-hosted specifications do not fully follow all the the required organization of printed Ecma standards. To conform to the Ecma print structure new HTML markup needs to be added to the ecmarkup generated HTML file and some sections need to be relocated within the document.

Ecmarkup makes extensive use of custom HTML tags. For example,`<emu-clause>`, `<emu-table>`, `<emu-alg>`, etc. Paged.js 0.4.1, used for the 2023 standards, generally worked fine with all such tags. They don't need to be removed or converted to `<div>`'s as was done in 2022.

The HTML for Ecma PDF generation is organized into four parts using `<section>` tags. Each `<section>` has a unique CSS `class`. They are: `cover`, `front-matter`, `clauses`, and `annexes`. The section classes are used for things like selecting page number formatting, page placement (left or right pages), etc. The organization of the sections are:

- cover (no page numbers)
  - front cover: title and Ecma logo
  - inside cover: Ecma contact info
- front-matter (starts on first "right side" (odd numbered) page). Roman page numbers
  - Ecma copyright notice and license (but not the software license). Should be one page. Page number: i
  - Blank page (unless copyright spills to two pages)
  - Print TOC (starts on right side page, a place holder until the actual table of contents is created)
  - Introduction (start right side)
    - Ends with General Assembly approval statement
  - Metadata page(s)—TC39 contact info, etc. (page break after Introduction)
- clauses (start right side, pages numbered starting at 1)

  - All of the `<emu-clause>` elements and their content. The first clause is immediately preceded by the title of the standard. Note that with the Ecma document format, only the title/first clause starts on a new page.
- annexes (start right side)
  - Annex 1
  - Annex 2 (start right side)
  - Additional annexes (each start right side)
  - Bibliography (if the document has one; start right side)
  - Software License (start right side)
  - Colophon (start right side)

The cover section serves as a place holder for two pages, the printed front cover and the inside front cover. The content in the HTML file isn't used in the final PDF. Instead, it will be replaced by normal Ecma document cover and Ecma contact info on the inside front cover. PDFs for the official cover and inside cover can be created for you by the Ecma secretariat and then edited into the paged.js generated PDF. (*Warning*: Don't use Mac Preview to do this—it will break all internal hyperlinks)

The Print TOC is a place holder. The actual creation of the TOC is covered in a subsequent step of this process.

The title headings of each formal annex ultimately needs to be updated to conform to the Ecma publication format. That is covered later in this document. The Software License and Colophon are technically not part of the standard, instead they are meta information about the standard. But for simplicity `<emu-annex>` can be used for their definition. The Software License `<emu-annex>` should not include the copyright license or ECMAaddress paragraphs. The software license text content must be moved from the Ecma copyright page of the front-matter.

Expressed in HTML, here is the document body organization:

```html
<body>
  <!-- deleted interactive screen sidebar was here-->
  <dev class="spec-container"> <!-- ecmarkup generated-->
    <section class="cover printonly">
      <!-- placeholder title and logo image goes here-->
      <div class="page-break"></div>
      <p class="ECMAaddress">
      <!-- etc. -->
    </section>
    <section class="front-matter">
      <div class="front-copyright-block page-break-right" >
        <!-- copyright text and copyright license elements go here -->
      </div>
    <div class="page-break">&nbsp;</div> <!-- something on blank page to force paged.js shou
    <div id="print-toc" class="print-toc page-break-right">
      <h1 class="emu-intro print-toc"><span>Contents</span><span>Page</span></h1>
      <!-- This is where the items of the print Table of Content get inserted.-->
    </div>
    <emu-intro id="sec-intro">
        <h1>Introduction</h1>
        <!-- Introduction elements go here -->
    </emu-intro>
    <p>This Ecma Standard ... adopted by the General Assembly of June <span class=year></spa
    <div class="page-break">
      <div id="metadata-block">
        <!-- metadata block elements go here -->
      </div>
    </div>
  </section>
  <section class="clauses">
    <h1 class="std-title">***Place title of standard here***</h1>
    <emu-clause id="scope">
        <h1><span class="secnum">2</span> Scope</h1>
        <!-- other elements of this clause go here -->
    </emu-clause>
    <emu-clause id="conformance">
        < h1><span class="secnum">2</span> Conformance</h1>
        <!-- other elements of this clause go here -->
    </emu-clause>
    <!-- all the remaining <emu-clause> follow here -->
  </section>
  <section class="annexes">
    <emu-annex id="**annex id goes here**">
      <div class="annex-title printonly">
        <h1>Annex A</h1>
        <h1 style="font-weight: normal;">(informative)</h1>
        <h1>***annex title goes here***</h1>
      </div>
      <!-- other elements of this annex go here -->
    </emu-annex>
    <!-- remaining <emu-annex for annexes with letter secnums follow here -->
    <emu-annex id="sec-bibliography">
      <div class="annex-title"><h1>Bibliography</h1></div>
      <!-- Bibliography elements go here -->
    </emu-annex>
    <emu-annex id="sec-copyright-and-software-license"  class="page-break-right">
      <div class="annex-title"><h1>Software License</h1></div>
      <!-- the software license paragraphs -->
    </emu-annex>
    <emu-annex id="sec-colophon" class="page-break-right">
```

```
      <div class="annex-title"><h1>Colophon</h1></div>
      <!-- the Colophon paragraphs -->
    </emu-annex>
  </section>
<div class="print-only page-break">&nbsp</div>
</div> <!-- spec-container-->
</body>
```

Currently these changes need to be made manually, but it is easy to imagine a future ecmarkup option for generating a HTML file with these changes.

If a new edition of a specification only has a relatively small number of changes from the previous edition going through this reorganization process may be unnecessary. Consider using a copy of the print HTML file for the previous edition as a starting point and copy the changes in the new edition into the copy.

Eliminate or disable web ui related tags and content

Near the front of the `<body>` there will be HTML tags and content relating to the web UI for browsing the specification document. This includes the interactive table of contents. All of this should be commented out or deleted. If there is other interactive web UI content within the HTML file it should also be deleted.

Alternatively, you can use the CSS class `webonly` (defined in `ecmarkup-ecmapdf-20XX.css`) to cause these tags to be ignore when printing. Either add the `webonly` class to a tag or wrap a `<div class="webonly">` around them.

For a large document such as ECMA-262, it is probably better to delete these interactive UI tags and their content. This may improve paged.js performance and editing ease.

Step 4 Get Paged.js working with the document

You're now ready to make an initial attempt at getting Paged.js to paginate your document. Generic instructions for using Paged.js are at https://pagedjs.org/documentation/2-getting-started-with-paged.js/. Unless you want to mess with browser CORS settings you need to place the html file and all locally referenced resources such as css and image files on a web server. Paged.js claims to work best with Chromium-based browsers, but you may have success with browsers that use other engines.

Open the HTML document in your browser, make the browser window as large as possible. You may initially see an unstyled presentation of the document flash by, but in a few seconds you should see a book-like 2-pages side-by-side paginated presentation of the document. If your document is very large (for example, ECMA-262) it may make several minutes to fully paginate the document. But, after the initial pages appear, if your scroll down to the end of the document you should see a flow of processed pages being added to the end of the document. If all goes well, when this flow stops the last pages of your document will be displayed. If so, you can move on to Step 5 of this process. But there can be issues that cause Paged.js to stop processing before reaching the end.

Debugging paged.js processing of ecmarkup HTML documents

Paged.js 0.4.1 generally does not have problems processing through ecmarkup custom elements. But occasional problems have been encountered with `<emu-import>` custom elements. Note that `<emu-import>` elements will have already been expanded during ecmarkup HTML generation so they don't serve any purpose during Paged.js processing. If a`<emu-import>` seems to be causing problems delete or comment out the opening and matching closing `<emu-import>` tags.

Paged.js works by first examining the DOM of the document to make an initial determination of where page breaks are going to appear. It then sets up a container `<div>` for each page and then relocates the original document elements into those page size containers.

You can observe the results of this process by using your browser's DOM inspector.

The most common problems occur because the elements that Paged.js predicted would fit on a page don't actually fit. For ecmarkup generated documents, this is usually caused by long tables or `<ecu-alg>` or `<ecu-grammar>` custom elements with many deeply nested child elements. Typically this does not cause Paged.js to stop processing document. But it can cause issues such as incorrectly rendered tables, truncated or missing text, unnecessary page breaks, or the insertion of overflow pages.

Step 5 Manual Cleanup

Once Paged.js is able to process through the entire HTML document you sill need to review the results and manually correct any formatting problems. You don't need to print the resulting document to review it—you can review it in the browser. Many issues can be corrected by adding a CSS class or `style` attribute to tags within the ecmarkup generated HTML. Some issues involve more extensive editing of the generated HTML.

`ecmarkup-ecmapdf-20XX.css` defines a few convenience class you can use:

class	CSS property	Notes
page-break	`break-before: page;`	page break before this element
page-break-right	`break-before: right;`	Start this element on a new right side page

class	CSS property	Notes
dont-break	`break-inside: avoid;`	Try to not insert page breaks within this element
dont-break-after	`break-after: avoid;`	Try to not page break immediately after this element
dont-justify	`text-align: left;`	change text alignment within this element
justify	`text-align: justify;`	change text alignment within this element
no-top-margin	`margin-top: 0;`	
no-bottom-margin	`margin-bottom: 0;`	
no-top-padding	`padding-top: 0;`	
no-bottom-padding	`padding-bottom: 0;`	

One time fixups

Formatting issues that don't involve pagination typically only need to be fixed once while cleaning up a document. Issues to look for include:

- Text that should be justified but isn't

  Printed Ecma standards are supposed to have justified text. `ecmarkup-ecmapdf-20XX.css` sets paragraphs to `text-align: justify;` but some blocks of text may be in other kinds HTML elements that are not defaulting to `justify`. Set the class of such elements to `justify`.

- Blocky text justification

  Long unbreakable text spans, such as long camel case names may make some paragraphs impossible to fully justify. Set such paragraphs to class `dont-justify`.

- Bad justification of last line of a paragraph

  Sometimes the browser text justification tries to justify a short final line of a paragraph. If you can't figure out why—set the paragraph to class `dont-justify`.

- Unusually large spacing between elements.

  This is often cause by one of the elements being something other than a `<p>`. Check top or bottom padding and margin of the adjacent elements and explicitly set as necessary to get uniform spacing.

  The most common such situation should already be handled by the rules in `ecmarkup-ecmapdf-20XX.css`. If you run into any new ones you may want to add an additional rule to the CSS file. For example:

```
emu-note > div.note-contents > emu-alg > ol:last-of-type {
  margin-bottom: 0;
}
```

- External hyperlinks (links to websites, documents, etc) whose URL should be visible in a printed document.

  A reader of a printed document can't click on a hyperlink to view the referenced content. Instead they have to type a URL into a browser. So make sure the URLs for external links are visible.

- Review the print rendering of all `<pre>` tags

  They are probably fine but occasionally an issue is present.

- Check for tables that are wider than a print page

  Ecmarkup documents that contain tables (or perhaps other elements) that are wider than the expected view port are typically styled such that they can be horizontally scrolled: `overflow-y: scroll`. If needed, the appropriate document specific CSS rules are typically defined in a `<style>` element in the document `<head>`. If there are any such rules they should be edited to be conditional on a `@media screen` media query. This will likely cause such wide elements to be truncated on the right side when printed. Making such elements fit horizontally may require some editorial changes.

  So far, in TC39 specifications the only occurrences of wide tables is in clause 16.2.1.5.4 of ECMA-262. This clause has a number of tables which are wide but short (too many wide unbreakable column labels but only a few short row labels). The overflow problem was fixed by pivoting the tables (exchanging rows and columns). This makes the tables fit horizontally but doesn't change their meaning or readability. If you are converting a new edition of ECMA-262 you should propagate forward the HTML for the pivoted tables from the previous edition's Paged.js HTML version.
  Be sure to check for any changes to the tables content for the new edition.

Pagination fixups

The clauses and subclasses of an Ecma specification are supposed to be formatted as a continuous stream of text divided into to page size units. Only each annex and parts of the front matter are explicitly specified as starting on a new page.

Within the clauses of a printed Ecma specification there should not be any pages with large blocks of whitespace at their bottom. A useful heuristic is that if more than 20% of a page is bottom whitespace there is something wrong with its pagination. Within this guide the term "premature break" is used to describe this problem.

Other pages have the opposite problem—too many lines of text has been included on a page. The document would look and read better if the page break can a few lines soon leaving a a bit more whitespace at the bottom of the page. Within this guide the lines that should move to the next page are called “orphans”.

In fixing pagination issues it is important to work forward from the front of the document to the back. Any pagination change is likely to invalidate any explicitly pagination that has already been inserted further on in the document. The process of working from front to back reviewing/fixing pagination is tedious and time consuming. You don’t want to have to redo it very many times.

The rest of this section is a survey of the most common pagination issues that show up when Paged.js processes a restructured ecmarkup-generated HTML document and how to fix them. Other issues that may arise can likely be fixed using some variation of these approaches.

- Orphan Section Headings

  Generally a clause or subclause title should be on the same page as the first paragraph of the section and the `ecmarkup-ecmapdf-20XX.css` rules specify to `break-after: avoid` for most heading elements. But occasionally Paged.js generates an orphaned section heading. Fix those by adding the class `page-break` to the corresponding `<emu-clause>` tag.

- When Paged.js page breaks an equation the first line on the new page may have the wrong margin. Need to unset left margin

- Tables pagination issues

  - Orphan Table Headers

    If a table starts near the bottom of a page sometimes the headings row (or perhaps the headings and a single data row) will be “orphaned” at the very bottom of the page with all of the rest of the table on the next page. This typically looks ugly. The fix is to force a page break immediately before the table. This can be done by adding the “page-break” class: `<emu-table class="page-break">`.

    A similar techniques can be used with `<emu-note>`, `<emu-alg>`, `<emu-clause>`, `<emu-grammar>`, etc. that are leaving something orphaned at the bottom of a page.

  - Premature Break before Table

    Sometimes Paged.js will start a table on a new page, leaving a large whitespace block on the preceding page. This typically happens when the table is too big to completely fit on the initial page. This is an indication that Paged.js couldn’t figure out how to split the table across the two pages.

To fix this, estimate at which table row the page break should occurs if it started on the initial page. Add the “page-break” class to the row that should come after the desired page break: `<tr class="page-break">`.

– Page Break within a multiline table row

  Table rows that have multiline content (or a complex structure such as containing a nested table) may be split somewhere on an interior line of the row. This is usually ugly or confusing. Fix it by adding the “page-break” class to the `<tr>` of the row. If that doesn’t work the original will have to be restructured as multiple tables—one for each page.

– Multipage Tables Needs a heading and caption on each page.

  The Ecma document styling guidelines requires that a formal Table (one with a table number and caption) that spans multiple pages must have a heading row and and caption on each page. After the first page, the caption is a continuation caption. Simply adding a “page-break” class to a `<tr>` doesn’t meet those requirements. Instead the original table needs to be split into multiple tables—one per page.

  Assume that Table 1 is a single column table needs to be split immediately before a row containing “content 2” defined as:

```
<tr><td>content 1</td></tr>
<tr><td>content 2</td></tr>
```

  The table can be split by adding the following tags:

```
...
<tr><td>content 1</td></tr>
<!-- terminate the original table-->
  </tbody></table></figure></emu-table>
<!-- replicate the original table heading with a page break-->
  <emu-table class="page-break">
    <figure><figcaption class="continued">Table 1</figcaption>
    <span id="table-1-part2"></span> <!-- added table needs new id -->
    <table>
       <tbody>
       <tr> <!--duplicate column headings from original table-->
         <th>Column Heading</th>
       </tr>
<!-- end prologue of second page table -->
<tr><td>content 2</td></tr>
...
```

  Tables that span more than two pages will need to be split multiple times.

- `<emu-note>` contents too wide after page break.

  Paged.js usually correctly inserts page breaks when the "note-contents" of an `<emu-note>` overflows a page. However, when this happens the content area on the second page is wider than on the first page. This happens because Paged.js on the second page is failing to provide the left "column" that on the first page contains "NOTE" (or "NOTE 2" etc.)

  This can be fixed by splitting the `<emu-note>` into parts at the point where the page break should occur. Assume we have the following emu-note and that Paged.js is inserting a page break between the two paragraphs.

  ```
  <emu-note><span class="note">Note 1</span><div class="note-contents">
    <p>bla-bla-bla 1...</p>
    <p>bla-bla-bla 2...</p>
  </div></emu-note>
  ```

  The second paragraph will show up on the second page and its content will be shifted to the left relative to the first paragraph. We fix this by creating a second emu-note for the second paragraph:

  ```
  <emu-note><span class="note">Note 1</span><div class="note-contents">
    <p>bla-bla-bla 1...</p>
  </div></emu-note> <!-- copied from end of original emu-note -->
  <!-- continue with new note on new page-->
  <emu-note class="page-break"><span class="note">&nbsp;</span>
    <div class="note-contents">
    <p>bla-bla-bla 2...</p>
  </div></emu-note>
  ```

  The `<span>` that is the first nested element of a `<emu-note>` contains the NOTE label that displays to the top left of the note contents. The second page `<emu-note>` intentionally does not display a NOTE label and the `<span>` only contains a non-breaking space. The `&nbsp;` is necessary for the correct placement of the note contents.

- `<emu-alg>` pagination issues

  Paged.js generally does a pretty good job paginating ecmarkdown algorithms. However, there are issues that arise. They are usually pretty obvious to spot when inspecting the print ready rendering.

  – Orphaned step 1 If an algorithm starts very near the bottom of a page Paged.js may decide insert a page break imediately after the first step. It often looks better for the page break is before the first step. Especially if the first step is follow by an indented substep. Force the first step to a new page by adding `page-break` to the `<emu-alg>` `class` attribute.

    Eliminating orphaned first-steps using this technique arguably improves the page layout and readability of the algorithm. But the improvement is relatively minor and the manually placed page break likely will need to be changed every time an edit earlier in the document changes pagination. For large documents (e.g. ECMA-262) that can be a lot of work. It's a judgement call (both for individual cases and overall) whether eliminating the orphaning of an algorithm's first step is worth the effort.

– Premature Break before a `<emu-alg>`

  Sometimes Paged.js will start an algorithm on a new page, leaving a large whitespace block on the preceding page. This typically happens when the algorithm is too long to completely fit on the page initial page. This is an indication that Paged.js couldn't figure out how to split the algorithm across the two pages. If is most likely to occur with very long algorithms with some deeply nested steps.

  To fix this, estimate at which line of the algorithm the page break should precede if the algorithm started on the initial page. Add the `page-break` class to the algorithm's list item that should come after the page break: `<li class="page-break">`. It's fine to try this with a algorithm substep (a nested list item) but that often doesn't work. Top level steps seem to work best.

  It's usually best to initially underestimate how many stop will fit on the initial page. If that works and there is still available whitespace on the initial page try move the break forward as much as will work.

– Premature Break not fixed by adding `page-break` to a list item

  Sometimes simply adding a `page-break` to an algorithm's list item doesn't fix a premature break issue. In that case the `emu-alg` needs to be split into two (or more) `emu-alg` similarly to what is described above for `emu-table`. But this process is more complicated because care must be taken to preserve the substep nesting and step numbering across the split.

  Assume you have an `emu-alg` that is initially defined as:

```
<emu-alg><ol>
  <li><span aria-hidden="true">1. </span>step 1</li>
  <li><span aria-hidden="true">2. </span>step 2<ol>
    <li><span aria-hidden="true">a. </span>step 2.a<ol>
      <li><span aria-hidden="true">i. </span>step 2.a.i</li>
      </ol>
    <ol></li>
  <li><span aria-hidden="true">3. </span>step 3</li>
</ol></emu-alg>
```

  This can be split into two linked `emu-alg` with a page break before step 2.a.i by rewriting it as:

```html
<emu-alg><ol>
  <li><span aria-hidden="true">1. </span>step 1</li>
  <li><span aria-hidden="true">2. </span>step 2<ol>
    <li><span aria-hidden="true">a. </span>step 2.a<ol>
    <!-- needs an explicit page break after this step-->
  </ol></li></ol> <!--close all open li and ol -->
</emu-alg>

<emu-alg class="page-break">
  <!-- reestablish the nested structure -->
  <ol class="no-top-margin" start="2"><li style="list-style:none;">
    <ol start="1"><li style="list-style:none;"><ol>
      <!-- continuation of the split algorithm-->
      <li><span aria-hidden="true">i. </span>step 2.a.i</li>
      </ol>
    <ol></li>
  <li><span aria-hidden="true">3. </span>step 3</li>
</ol></emu-alg>
```

This same pattern can be extended to break at a deeper outline level or for algorithms that are too long to fit on two pages.

This example breaks before a third level step (2.a.i) If possible, it's slightly simpler and probably preferable to break at a first or second level step.

If you picked a good place to break, the `class="page-break"` on the second `<emu-alg>` is usually unnecessary as Paged.js will decide that the end of the first `<emu-alg>` is an appropriate place for a page break.

Improving future editions

With each edition of the TC39 standards, Paged.js has improved its reliability and its support of CSS Paged Media features. So has our understanding of how to use Paged Media features. There are plenty of ways we might improve the process in the future. Here are some possibilities to consider:

- Each year check for and use new Paged.js versions with bug fixes and more complete CSS Paged Media support.
- Get figure 1 to fit across the page when using the SVG image
- Experiment with using some of the more sophisticated CSS paragraph line layout features to get better text justification
  - try text hyphenation
  - experiment with using the CSS `widows` and `orphans` properties.
- Add PDF bookmark properties to section headings
- The current CSS for the table of contents requires a lot of manual tweaking of the TOC entries. Use more sophisticated CSS layout techniques to

eliminate the need of all the tweaking.

- Explore using CSS Paged Media techniques to generate the actual TOC entries.
- Instead of editing pagination or style attributes directly into document HTML tags, consider using an ID attribute (or a custom attribute) to identify the tags that need additional print specific attributes and place all the additions in a separate CSS file. This may make it easier to carry forward such attribute placement from edition to edition.
- Experiment with eliminating the need to manually format long tables that span page breaks. In theory, Paged.js should replicate tables headings defined inside `<thead>` elements when it needs to page break a table. But that doesn't work with Paged.js 0.4.1—but it may work in a follow-on version. Even if that works it will still be necessary to insert Table continues captions. Perhaps there are CSS tricks for generating them.
- Eliminate the need to manually restructuring the screen HTML file generated by ecmarkup into the print structure HTML file. Instead, enhance ecmarkup with an options to generate a print format HTML document that follows the document structure described in this guide.