ADMINISTRATIVE GUIDELINE

HTML Pub — Tooling and documentation for HTML documents

Approved: 2026-05-19

Table of contents

Foreword

The Society of Motion Picture and Television Engineers (SMPTE) is an internationally-recognized standards developing organization. Headquartered and incorporated in the United States of America, SMPTE has members in over 80 countries on six continents. SMPTE's Engineering Documents, including Standards, Recommended Practices, and Engineering Guidelines, are prepared by SMPTE's Technology Committees. Participation in these Committees is open to all with a bona fide interest in their work. SMPTE cooperates closely with other standards-developing organizations, including ISO, IEC and ITU. SMPTE Engineering Documents are drafted in accordance with the rules given in its Standards Operations Manual.

For more information, please visit www.smpte.org.

This Standards Administrative Guideline forms an adjunct to the use and interpretation of the SMPTE Standards Operations Manual. In the event of a conflict, the Operations Manual shall prevail.

Introduction

Authoring SMPTE documents consist of two aspects:

- structure and syntax of the document; and
- managing revisions using GitHub.

1 Scope

This Administrative Guideline describes the process for authoring SMPTE documents using HTML.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

HTML Standard, Living Standard
url: https://html.spec.whatwg.org/multipage/

SMPTE Standards Operations Manual
url: https://doc.smpte-doc.org/standards-operations-manual/main/

SMPTE AG 02, Document Naming and Packaging
url: https://doc.smpte-doc.org/ag-02/main/

SMPTE AG 07, Document Flowcharts
url: https://doc.smpte-doc.org/ag-07/main/

SMPTE AG 31, GitHub Operating Guidelines
url: https://doc.smpte-doc.org/ag-31/main/

SMPTE AG 33, Document Process and Workflow
url: https://doc.smpte-doc.org/ag-33/main/

3 Terms and definitions

For the purposes of this document, the terms and definitions given in the following documents apply:

- SMPTE Standards Operations Manual
- SMPTE AG 31

4 Conformance

The following keywords have a specific meaning in the context of this document:

- *shall* and *shall not* express a requirement from which no deviation is permitted;
- *should* and *should not* express a strong recommendation without necessarily mentioning or excluding other choices;
- *may* expresses explicit liberty (or opportunity) to do something;
- *Note* and *informative* indicates that the associated prose is not indispensable, and can be removed, changed, or added editorially without affecting the scope, or the document's usage.

5 Quick start

1. Install the git source control system
2. Install a text editor and local web server. Visual Studio Code (https://code.visualstudio.com/) combined with the `ms-vscode.live-server` extension (https://marketplace.visualstudio.com/items?itemName=ms-vscode.live-server) is one option.
3. Clone the HTML publication template:

   ```
   git clone --recurse-submodules https://github.com/SMPTE/html-pub-template.git
   ```

4. (Advanced) Add the pre-commit hook specified at 8.1.7 by following the instructions at https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks.
5. If working on a project that has a repository, update the `origin` remote to that repository:

   ```
   git remote set-url origin <HTTPS URL of the project repository found on GitHub>
   ```

   For example:

   ```
   git remote set-url origin https://github.com/SMPTE/stxxx-y-private.git
   ```

6. Start editing the main prose element at `doc/main.html`.

6 Directory layout

A document consists of multiple files arranged in the following directory structure:

- `doc/main.html`: main element of the document (see Clause 7).
- `doc/media/`: directory containing media referenced by the main element (see 7.4.9).
- `doc/elements/`: directory containing non-prose elements of the document (see 7.3.10).
- `doc/snippets/`: directory containing snippets embedded in the main element (see 7.4.11).
- `tooling/`: directory consisting of the tools necessary to render SMPTE documents (see Clause 8).
- `.smpte-build.json`: JSON file used to configure the HTML rendering process (see 8.1.3).
- `.gitignore`: Prevents build and operating system artifacts from being tracked by Git.
- `.github/workflows/main.yml`: Allows automated HTML rendering and redlining in GitHub (see 8.1.4).

7 Main element

7.1 General

The main element shall be an HTML document represented using the XML syntax, as specified at HTML Standard.

The basic structure of the main element is illustrated at Figure 1.

```
<!doctype html>
<html>
  <head itemscope="itemscope" itemtype="http://smpte.org/standards/documents">
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="ie=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <script type="module" src="../tooling/smpte.js"></script>
    <meta itemprop="pubType" content="AG" />
    <meta itemprop="pubNumber" content="XX" />
    <!-- <meta itemprop="pubPart" content="99" /> -->
    <!-- <meta itemprop="pubSuiteTitle" content="Suite title" /> -->
    <meta itemprop="pubState" content="draft" />
    <meta itemprop="pubDateTime" content="20XX-XX-XX" />
    <!-- <meta itemprop="pubStage" content="WD" /> -->
    <!-- <meta itemprop="pubTC" content="27C" /> -->
    <!-- <meta itemprop="pubConfidential" content="no" /> -->
    <!-- <meta itemprop="pubRevisionOf" content="SMPTE ST XXXX-Y:ZZZZ" /> -->
    <title>Title of the document</title>
  </head>
  <body>
    <section id="sec-scope">
      <p>This is the scope of the document.<p>
    </section>
  </body>
</html>
```

Figure 1 — Basic structure of the main element.

The `smpte.js` script contains the code that renders the HTML document provided by the author into an HTML document appropriate for publication. This rendering happens automatically when loading the document in a web browser.

7.2 `head` element

7.2.1 General

The `head` element shall contain the following:

- `itemscope` attribute equal to `itemscope`;
- `itemtype` attribute equal to `http://smpte.org/standards/documents`;
- `<meta charset="utf-8" />`;
- `<meta http-equiv="x-ua-compatible" content="ie=edge" />`;
- `<meta name="viewport" content="width=device-width, initial-scale=1" />`
- a `script` element whose `src` attribute links to the `smpte.js` module in the `tooling` directory.

The `title` element shall be equal to the title of the document, excluding:

- the name of the publisher, e.g., SMPTE;
- the document type, e.g., ST;
- the document number, e.g., 428-1;
- the title of the suite if the document is part of a suite.

EXAMPLE

The `title` of SMPTE ST 429-2 is *DCP Operational Constraints*.

In addition to the `title` element, the `head` element contains SMPTE publication metadata in the form of `meta` elements as specified below.

7.2.2 pubType

The `head` element shall contain exactly one `meta` element with an `itemprop` attribute equal to `pubType`.

The element shall have a `content` attribute equal to one of the following values:

- `ST`, if the document is a Standard;
- `RP`, if the document is a Recommended Practice;
- `EG`, if the document is an Engineering Guideline;
- `ER`, if the document is an Engineering Report;
- `RDD`, if the document is an Registered Disclosure Document;
- `AG`, if the document is an Administrative Guideline; or
- `OM`, if the document is an Operations Manual.

7.2.3 pubNumber

The `head` element shall contain zero or one `meta` element with an `itemprop` attribute equal to `pubNumber`.

The element shall be present if:

- `pubState` is `pub`; or
- `pubPart` is specified; or
- `pubVersion` is specified.

The `content` attribute of the element shall be equal to the document number, which is a sequence of digits.

7.2.4 pubPart

The `head` element shall contain exactly one `meta` element with an `itemprop` attribute equal to `pubPart` if the document is part of a suite, and none otherwise.

The element shall have a `content` attribute equal to the document part number, which is a sequence of digits.

7.2.5 pubSuiteTitle

The `head` element shall contain exactly one `meta` element with an `itemprop` attribute equal to `pubSuiteTitle` if the document is part of a suite, and none otherwise.

The element shall have a `content` attribute equal to the title of the document suite.

7.2.6 pubVersion

The `head` element shall contain zero or one `meta` element with an `itemprop` attribute equal to `pubVersion`.

The element shall have a `content` attribute. The value of this attribute overrides the document version that is automatically generated by the tooling for Engineering Document and the `pubStage` is `PUB`.

This element shall be present if and only if the document version does not follow current practice.

7.2.7 `pubState`

The `head` element shall contain one `meta` element with an `itemprop` attribute equal to `pubState`.

The element shall have a `content` attribute equal to one of the following values:

- `draft`, if the document is a draft.
- `pub`, if the document is published.

For further explanation of determining the value of the document's `pubState`, see SMPTE AG 33

7.2.8 `pubStage`

The `head` element shall contain zero or one `meta` element with an `itemprop` attribute equal to `pubStage`.

The element shall be present if the document is an Engineering Document.

The element shall have a `content` attribute equal to one of the following values:

- `WD`, if the document is a Working Draft;
- `CD`, if the document is a Committee Draft;
- `FCD`, if the document is a Final Committee Draft;
- `DP`, if the document is a Draft Publication; or
- `PUB`, if the document is a Published.

For further explanation of determining the value of the document's `pubStage`, see SMPTE AG 33

7.2.9 `pubRevisionOf`

The `head` element shall contain zero or one `meta` element with an `itemprop` attribute equal to `pubRevisionOf`.

The element shall have a `content` attribute equal to the identifier of the document that the document revises.

7.2.10 `pubTC`

The `head` element shall contain zero or one `meta` element with an `itemprop` attribute equal to `pubTC`.

The element shall be present if the document is an Engineering Document.

The element shall have a `content` attribute equal to the identifier of the TC responsible for the document.

7.2.11 `pubConfidential`

The `head` element shall contain zero or one `meta` element with an `itemprop` attribute equal to `pubConfidential`.

If present, the element shall have a `content` attribute equal to one of the following values:

yes

The Engineering Document is SMPTE confidential.

no

The Engineering Document is not SMPTE confidential.

The element shall not be present unless the document is an Engineering Document.

Unless specified otherwise, an Engineering Document that is not at publication stage is confidential.

7.2.12 `pubDateTime`

The `head` element shall contain zero or one `meta` element with an `itemprop` attribute equal to `pubDateTime`.

The element shall be present if `pubState` is `pub`.

The element shall have a `content` attribute that conforms to the pattern `YYYY-MM-DD`, where `YYYY`, `MM` and `DD` are the year, month and day of the most recent approval date of the document.

See AG-02 for a definition of approval date.

7.2.13 `effectiveDateTime`

The `head` element shall contain zero or one `meta` element with an `itemprop` attribute equal to `effectiveDateTime`.

The element shall be present if the document is an Operations Manual and the `pubState` is `pub`.

The element shall have a `content` attribute that conforms to the pattern `YYYY-MM-DD`, where `YYYY`, `MM` and `DD` are the year, month and day that the document becomes effective.

7.2.14 Additional `script` elements

The `head` element may contain additional `script` elements that import JavaScript libraries for document layout, e.g. MathJax

EXAMPLE

```
<script src="https://polyfill.io/v3/polyfill.min.js?features=es6"></script>
<script type="text/javascript" id="MathJax-script"
  async src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-mml-svg.js">
</script>
```

7.2.15 Additional `stylesheet` and `style` elements

The `head` element shall not contain additional `link` elements that import either internal or external `stylesheet` files, nor shall it contain inline `style` elements.

EXAMPLE

```
<link rel="stylesheet"  href="./styles.css">
<style>
  p { color: red; }
</style>
```

NOTE — Also see 7.4.19 for guidance on styling.

7.3 `body` element

7.3.1 Overall structure

The `body` element shall consist of an ordered sequence of `section` elements to create their respective clauses, as specified at Table 1.

Table 1 — Ordered sequence of clause / `section` elements comprising the `body` element.

Clause name / `section` element	Cardinality
Foreword	0 or 1
Introduction	0 or 1
Scope	Exactly 1
Normative references	0 or 1
Terms and definitions	0 or 1
Conformance	0 or 1
Prose	0 or more
Annex	0 or more
Additional elements	0 or 1
Bibliography	0 or 1

From the author-supplied `section` elements specified at Table 1, the tooling generates a document structure that conforms to SMPTE editorial requirements. For example:

- clauses are numbered automatically;
- headings of `section` elements (clause names) other than Prose and Annex are automatically generated;
- boilerplate is added, including the SMPTE masthead.

7.3.2 Foreword `section`

7.3.2.1 General

The Foreword `section` contains author-supplied prose, e.g. list of substantive changes since the last edition, that is added to the SMPTE boilerplate text.

The absence of this `section` indicates that no author-supplied prose was provided.

The `id` attribute shall be present on the `section` element and equal to `sec-foreword`.

The heading shall not be present.

EXAMPLE

```
<section id="sec-foreword">
  <p>This is the additional information relevant to the document.</p>
```

```
</section>
```

7.3.2.2 RDD

When `pubType` is set to `RDD`, the Foreword `section` shall be present and there there shall be a single `dl`, after any author-supplied prose if supplied, which contains contact information about the proponent(s) of the RDD document.

The `id` attribute shall be present on the `dl` element and equal to `element-proponent`.

If present, the `dl` element shall use a single `dt` element for each company name designated as a Proponent of the document, and multiple lines of `dd` as needed for contact information for each company listed. The contact infomation should at least include an address and email, reach out to SMPTE HO for further deatails.

EXAMPLE

```
<section id="sec-foreword">
  <p>This is the additional information relevant to the document.</p>
  <dl id="element-proponent">
    <dt>Company 1 Name Here</dt>
      <dd>Contact Name</dd>
      <dd>Company Address</dd>
      <dd><a>Contact Email</a></dd>
      <dd><a>Company Website</a></dd>
    <dt>Company 2 Name Here</dt>
      <dd>Contact Name</dd>
      <dd>Company Address</dd>
      <dd><a>Contact Email</a></dd>
      <dd><a>Company Website</a></dd>
  <dl>
</section>
```

7.3.3 Introduction `section`

The Introduction `section` contains author-supplied prose that forms the introduction of the document.

The absence of this `section` indicates that the document has no introduction.

The `id` attribute shall be present on the `section` element and equal to `sec-introduction`.

The heading shall not be present.

EXAMPLE

```
<section id="sec-introduction">
  <p>This is the introduction.</p>
</section>
```

7.3.4 Scope `section`

The Scope `section` contains author-supplied prose that forms the scope of the document.

The `id` attribute shall be present on the `section` element and equal to `sec-scope`.

The heading shall not be present.

EXAMPLE

```
<section id="sec-scope">
  <p>This is the scope.</p>
```

```
</section>
```

7.3.5 Normative references `section`

The Normative references `section` collects normative references cited in the document.

The `id` attribute shall be present on the `section` element and equal to `sec-normative-references`.

The heading shall not be present.

The absence of this `section` indicates that no normative references are cited by the document.

If present, the `section` shall contain a single `ul` element where each `li` element is a normative reference, such that:

- Each `li` element shall contain one `cite` element, with value that contains the text that will be used when citing the reference in the prose.
- The `cite` element shall contain an `id` attribute with value prefixed by `bib-`.
- Each `li` element may contain additional title information.
- Each `li` element may contain one `a` element that contains a location where the reference can be retrieved.
- Each `li` element may contain one `span` element with a `class="doi"` attribute that contains the DOI of the document. Using this will also create a DOI resolver URL.

The DOI `span` element should be used, and not the `a` element, for both the most recent and specific editions of a SMPTE document. SMPTE HO can provide guidance for specific DOIs available.

Undated references should be used, since they point to the most up-to-date edition of a document.

Dated references shall be used when referencing a specific element of the document, e.g., a table. They should also be used when past and future editions of the document are likely to be inadequate.

EXAMPLE

```
<section id="sec-normative-references">
  <ul>
    <li>
      <cite id="bib-HTML-5">HTML Standard</cite>, Living Standard
      <a>https://html.spec.whatwg.org/multipage/</a>
    </li>
    <li>
      <cite id="bib-SMPTE-st429-18">SMPTE ST 429-18</cite>, D-Cinema Packaging - Immersive Audio Track
      File
      <span class="doi">10.5594/SMPTE.ST429-18</span>
    </li>
    <li>
      <cite id="bib-SMPTE-st429-18-2023">SMPTE ST 429-18:2023</cite>, D-Cinema Packaging - Immersive
      Audio Track File (2023 Edition)
      <span class="doi">10.5594/SMPTE.ST429-18.2023</span>
    </li>
  </ul>
</section>
```

7.3.6 Terms and definitions `section`

The Terms and definitions `section` collects terms, abbreviations and symbols that are not defined inline the clauses of the document, as described in 7.4.6.

The `id` attribute shall be present on the `section` element and equal to `sec-terms-and-definitions`.

The heading shall not be present.

The absence of this `section` indicates that all terms and abbreviations are defined inline the clauses of the document.

If present, the `section` contains one or both of the following elements:

- a `ul` element that cites external definitions, subject to the following constraints:
  - The `id` attribute of the `ul` element shall be equal to `terms-ext-defs`.
  - Each `li` element contains a citation link to a normative reference (see 7.3.5), whose definitions are included by reference.
- a `dl` element that contains individual definitions and abbreviations, subject to the following constraints:
  - The `id` attribute of the `dl` element shall be equal to `terms-int-defs`.
  - Each `dt` element contains a single term, abbreviation or symbol. Zero or more additional `dt` elements can follow the first `dt` element, in which case all the terms, abbreviations and symbols in the immediately preceding `dt` elements are synonyms.
  - The `dd` elements following the `dt` element(s) shall appear in the following order - each group is optional (except the definition), but if present, it shall appear in this sequence:
    1. **Deprecated:** the first `dd` element, if present, shall have a `class` attribute containing `deprecated`, indicating the term is deprecated. At most one such element is permitted.
    2. **Definition:** the next `dd` element shall be the definition of the term. This element is required. References to other defined terms and `dfn` elements are permitted using `<a>` elements without an `href` attribute. Citations to clauses or external references (i.e. `<a href="#bib-HTML-5">`) are not permitted.
    3. **Examples:** the next `dd` element(s), if present, shall each have a `class` attribute containing `example`.
    4. **Notes:** the next `dd` element(s), if present, shall each have a `class` attribute containing `note`. References to defined terms, clauses, and external references are all permitted.
    5. **Source:** the last `dd` element, if present, shall have a `class` attribute containing `source` and serves as the source of the definition. At most one such element is permitted. It shall contain a reference to a bibliographic entry and may contain additional text. If any changes are made to the original terminological entry, this shall be indicated, along with a description of what has been modified.

EXAMPLE

```
<section id="sec-terms-and-definitions">
  <ul id="terms-ext-defs">
    <li><a href="#bib-HTML-5"></a></li>
  </ul>
  <dl id="terms-int-defs">
    <dt><dfn>key number</dfn></dt>
    <dd class="deprecated">deprecated term</dd>
    <dd>number that is printed with ink or exposed onto the film at the time of
    manufacture at regular intervals, typically one foot</dd>
    <dd class="example">An example of a key number is 12345.</dd>
    <dd class="note">Key number shall not be confused with key frame.</dd>
    <dd class="source"><a href="#bib-key-number-spec"></a>, modified - definition has been updated.</dd>
  </dl>
</section>
```

NOTE — The boilerplate of the clause will be generated automatically.

7.3.7 Conformance `section`

The Conformance `section` contains author-supplied, conformance-related prose that is added to the SMPTE boilerplate text that is generated depending on `pubType`.

The Conformance `section` follows Terms and definitions and is the first SMPTE-defined clause that may carry substantive conformance requirements. It is positioned consistent with ISO/IEC Directives, Part 2, which reserve Clauses 1, 2, and 3 for Scope, Normative references, and Terms and definitions respectively.

The absence of this `section` indicates that no author-supplied, conformance-related prose is provided.

The `id` attribute shall be present on the `section` element and equal to `sec-conformance`.

The heading shall not be present.

EXAMPLE

```
<section id="sec-conformance">
  <p>These are additional conformance requirements and definitions.</p>
</section>
```

This `section` is prohibited unless the document is a Standard or Recommended Practice.

7.3.8 Prose `section`(s)

Each Prose `section` contains author-supplied technical prose.

A Prose `section` may contain other nested Prose `section` elements. If any child of a Prose `section` is a Prose `section`, then all children shall be Prose `section` elements.

The `id` attribute shall be present and its value should be prefixed with `sec-`.

The `class` attribute of a Prose `section` shall not contain the `annex` class.

A `section` heading element of the appropriate level shall be present, starting with level 2 (`h2`) for all Prose `section` elements that are children of the `body` element.

Prose `section` elements are automatically numbered, so neither the heading element nor its `id` attribute should contain numbering information.

EXAMPLE

```
<section id="sec-prose-clause">
  <h2>Prose Clause</h2>
  <section id="sec-prose-sub-clause">
    <h3>Prose Sub Clause</h3>
    <p>Some technical content</p>
  </section>
  <section id="sec-next-prose-sub-clause">
    <h3>Next Prose Sub Clause</h3>
    <p>Some technical content</p>
  </section>
</section>
```

7.3.9 Annex `section`(s)

An Annex `section` contains an author-supplied technical annex.

The requirements for an Annex `section` are the same as those of a Prose `section`, with the exception that the Annex `section` shall contain the `class` attribute with value of `annex`.

An Annex `section` shall not contain nested annex `class` elements.

By default, an Annex `section` is labeled as `(Normative)`. To label any Annex `section` as `(Informative)`, the `class` attribute value of `informative` shall be present.

EXAMPLE

```
<section class="annex informative" id="sec-additional-info">
  <h2>Additional Info</h2>
  <section id="sec-additional-info-sub-section">
    <h3>Additional Info Sub Clause</h3>
    <p>Some technical content</p>
  </section>
  <section id="sec-additional-info-sub-section-more">
    <h3>Additional Info Sub Clause More</h3>
    <p>Some technical content</p>
  </section>
</section>
```

7.3.10 Additional elements `section`

The additional elements `section` collects the non-prose elements of the document.

The `id` attribute shall be present on the `section` element and equal to `sec-elements`.

The heading of the `section` shall not be present.

If present, the `section` shall:

- come immediately before the Bibliography `section`; and
- contain a single ordered list `ol` where each element `li` contains exactly one `a` that links to an element of the document.

Each `a` element shall have:

- an `id` attribute whose value is prefixed with `element-`;
- a `title` attribute that provides a short description of the element;
- a `href` attribute that links to the element, as detailed below; and
- a `class` attribute that contains the token `informative` if the element is informative.

Each additional element shall be managed in one of the following ways:

- as a file in the `doc/elements` directory, in which case the `href` attribute shall be relative path to the file, e.g. `elements/element.txt` (Element a); or
- in a separate SMPTE repository, in which case the `href` attribute shall be the absolute URL to a version of the repo, e.g. `https://github.com/SMPTE/html-pub/tree/631ed7a0312fa183c356a7b742129ce46f92736f`

(Element b).

Non-SMPTE repositories and SMPTE repositories that are not an integral part of the document shall be cited using a bibliographic reference.

EXAMPLE

```
<section id="sec-elements">
  <ol>
    <li>
      <a id="element-sample-text" title="Description of the element" href="elements/form.docx"></a>
    </li>
  </ol>
</section>
```

NOTE — This clause will automatically be labeled as the last Annex clause and marked as informative.

7.3.11 Bibliography `section`

The Bibliography `section` contains author-supplied bibliographic references.

The `id` attribute shall be present on the `section` element and equal to `sec-bibliography`.

The heading shall not be present.

The absence of this `section` indicates that no bibliographic references are cited by the document.

If present, the `section` shall contain a single `ul` element that conforms to the same requirements as the `ul` element of the normative references `section`.

EXAMPLE

```
<section id="sec-bibliography">
  <ul>
    <li>
      <cite id="bib-iso-directives-part2">ISO/IEC Directives, Part 2</cite>,
      Principles and rules for the structure and drafting of ISO and IEC documents (Ninth edition, 2021)
      <a>https://www.iso.org/sites/directives/current/part2/index.xhtml</a>
    </li>
  </ul>
</section>
```

7.4 Other elements

7.4.1 Referencing clauses

When referencing a clause, an `a` element with its `href` attribute referencing a `section` element shall be used. The text of the link will be automatically set to clause number.

EXAMPLE

`<a href="#sec-scope"></a>` is rendered as Clause 1 and `<a href="#sec-section-linking"></a>` is rendered as 7.4.1

7.4.2 Citing normative references

When citing a normative reference, an `a` element with its `href` attribute referencing a `cite` element from the normative references `section` shall be used. The text of the link will be automatically set to the contents of the `cite` element.

EXAMPLE

`<a href="#bib-HTML-5"></a>` is rendered as HTML Standard.

7.4.3 Citing non-prose elements

When citing an non-prose element of the document, an `a` element with its `href` referencing an `a` element from the additional elements `section` shall be used. The text of the link will be automatically set to the letter of the element in element list.

EXAMPLE

`<a href="#element-sample-text"></a>` is rendered as Element a.

7.4.4 Citing bibliographic references

When citing a bibliographic reference in the prose, an `a` element with its `href` attribute referencing a `cite` element from the Bibliography `section` shall be used. The text of the link will be automatically set to the contents of the `cite` element.

EXAMPLE

`<a href="#bib-iso-directives-part2"></a>` is rendered as ISO/IEC Directives, Part 2.

7.4.5 External links

Links to external resources shall be written by wrapping the URL in an `a` element

EXAMPLE

`<a>https://www.iso.org/sites/directives/current/part2/index.xhtml</a>` is rendered as https://www.iso.org/sites/directives/current/part2/index.xhtml.

NOTE — An `href` attribute will automatically be generated.

7.4.6 Inline terms

A term can be defined by wrapping it in a `dfn` element.

EXAMPLE 1

```
<dfn data-lt="alternate1|alternate2">term</dfn>
```

The optional `data-lt` attribute specify alternate forms of the term, each separated by a `|` character.

An optional `id` attribute can be specified; otherwise one will be generated automatically.

EXAMPLE 2

The definition above can be referenced using one of the following forms:

- `<a>term</a>`, which is rendered as term
- `<a>alternate1</a>`, which is rendered as alternate1
- `<a>alternate2</a>`, which is rendered as alternate2

7.4.7 Lists of key-value pairs

A `dl` element can be used for key-value pairs lists.

Since styling is automatically applied to `dl` elements, author should not specify additional formatting, either in the form of CSS styles or using markup such as the `b` element.

At least one `dd` shall be present for each `dt` element.

EXAMPLE

```
<dl>
  <dt>imperative forms</dt>
  <dd>"see"</dd>
  <dt>non-imperative forms</dt>
  <dd>"according to"</dd>
  <dd>"as defined in"</dd>
  <dd>"as specified in"</dd>
  <dd>"details as given in"</dd>
  <dd>"in accordance with"</dd>
</dl>
```

is rendered as:

imperative forms

"see"

non-imperative forms

"according to"
"as defined in"
"as specified in"
"details as given in"
"in accordance with"

7.4.8 Lists

A list can be inserted using either an `ol` or `ul` element. The `ul` (unordered list) element should be used when the order of listed items is not meaningful or important. The `ol` (ordered list) element should be only used when the order of listed items is meaningful and/or important to the list and/or document.

General guidelines for lists:

- An `ol` or `ul` element shall contain one (1) or more `li` element(s).
- A `li` element may contain one (1) or more nested `ol` or `ul` element(s).
- An `ol` element should not be nested within an `ul` element.

EXAMPLE

```
<ol>
  <li>First list item</li>
  <li>Second list item</li>
</ol>
```

is rendered as:

1. First list item
2. Second list item

For more examples of various list layouts and optional list types, see Annex B

7.4.9 Figures

A figure can be inserted using a `figure` element, using the image format as defined in SMPTE AG 02, with the following requirements.

The `figure` element shall contain:

- an `id` attribute whose value starts with `figure-`
- one `img` or `pre` element
- one `figcaption` element following the `img` or `pre` element

If an `img` element is present, then its `src` attribute shall be present and be a relative path starting with the string `media/`.

Figures are automatically numbered.

EXAMPLE 1

```
<figure id="figure-sample-image">
  <img src="media/sample.svg">
  <figcaption>Example of a figure</figcaption>
</figure>
```

is rendered as

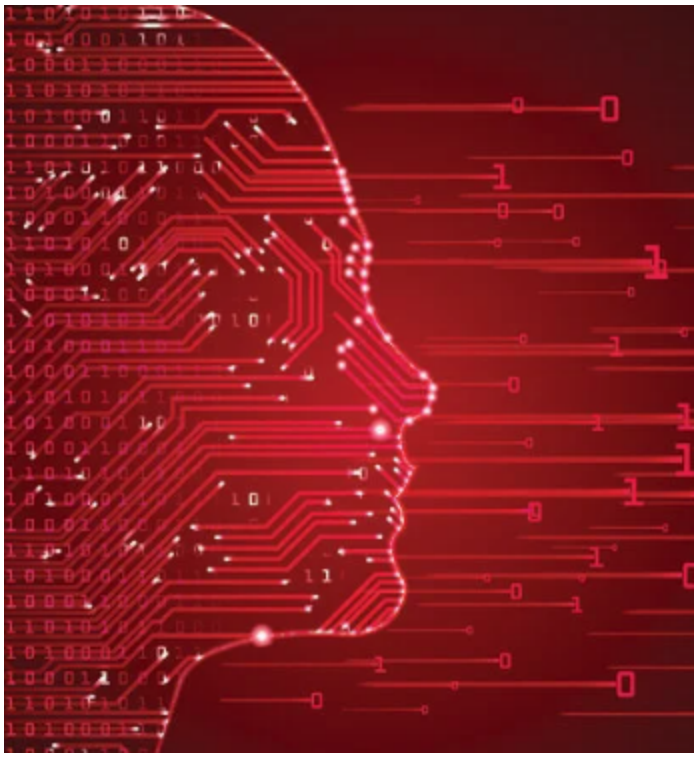

Figure 2 — Example of a figure

When referencing a figure in the prose, an `a` element with its `href` attribute referencing the `figure` element shall be used. The text of the link will be automatically set to the number of the figure.

EXAMPLE 2

`<a href="#figure-sample-image"></a>` is rendered as Figure 2.

7.4.10 Tables

Tables can be inserted by using a `table` element.

The `table` element shall contain:

- an `id` attribute whose value starts with `tab-`
- one `caption` element as the first child of the `table` element
- one `thead` element, containing one `tr` element, with `th` element(s) as needed for each column header
- one or more `tbody` element, with `tr` element(s) as needed for each row

Unless all columns are uniform in contents, the relative width of each column should be specified by setting the CSS `width` property as a percentage value on each `th` element.

NOTE 1 — Page breaks are avoided within a `tbody` element when generating PDF. Care should be taken for especially long tables to ensure rendering is as expected. For data in tables spanning multiple pages, other options should be considered and used.

EXAMPLE 1

```
<table id="tab-sample-tabledata">
  <caption>Sample Table</caption>
  <thead>
    <tr>
      <th>Sample Number</th>
      <th>Sample Name</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>0001</td>
      <td>Name 01</td>
    </tr>
    <tr>
      <td>0002</td>
      <td>Name 02</td>
    </tr>
  </tbody>
</table>
```

is rendered as

Table 2 — Sample Table

Sample Number	Sample Name
0001	Name 01
0002	Name 02

When referencing a table in the prose, an `a` element with its `href` attribute referencing the `table` element shall be used. The text of the link will be automatically set to the number of the table.

EXAMPLE 2

`<a href="#tab-sample-tabledata"></a>` is rendered as Table 2.

For more examples of various table layouts and optional cell/column text alignment, see Annex A

Tables may include footnotes. Footnotes shall appear in a `tfoot` element at the foot of the table, each as a `p` element with `class="footnote"` and a unique `id` attribute. Footnotes are automatically assigned superscript lowercase letters (a, b, c, etc) in the order they appear in the `tfoot` element. References to footnotes within the table body are marked with an `a` element whose `href` attribute references the footnote's `id`.

NOTE 2 — Footnotes are only permitted within tables. They must not be used outside of a `table` element.

NOTE 3 — Each footnote reference (`a` element) shall be in the same table as the footnote it references, and each footnote shall be referenced exactly once.

EXAMPLE 3

```
<table id="tab-sample-footnote">
  <caption>Sample Table with Footnotes</caption>
  <thead>
    <tr>
      <th>Sample Number</th>
      <th>Sample Name</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>0001</td>
      <td>Name 01 <a href="#fn-doc-example"></a></td>
    </tr>
    <tr>
      <td>0002</td>
      <td>Name 02</td>
    </tr>
  </tbody>
  <tfoot>
    <tr>
      <td colspan="2">
        <p class="footnote" id="fn-doc-example">Footnote content goes here.</p>
      </td>
    </tr>
  </tfoot>
</table>
```

is rendered as

Table 3 — Sample Table with Footnotes

Sample Number	Sample Name
0001	Name 01 [a]
0002	Name 02
[a] Footnote content goes here.	

Notes are permitted in tables cells or `tfoot` elements and are automatically numbered. When a note is also present in the `tfoot`, it shall appear before all footnotes:

EXAMPLE 4

```
<tfoot>
  <tr>
    <td colspan="2">
      <p class="note">Note content goes here.</p>
      <p class="footnote" id="fn-doc-example-2">Footnote content goes here.</p>
    </td>
  </tr>
</tfoot>
```

is rendered as

Table 4 — Sample Table with Note and Footnote in Footer

Sample Number	Sample Name
0001	Name 01 [a]

NOTE — Note content goes here.

[a] Footnote content goes here.

7.4.11 External code snippets

External code snippets can be inserted by setting the `data-include` attribute on a `pre` element to the relative path of the snippet.

EXAMPLE

```
<pre data-include="snippets/example.txt"></pre>
```

is rendered as

```
This an example.
```

Keeping code snippets separate from the main element allows the snippets to be individually formatted and validated.

External code snippets are not additional elements of the document and are not listed in the *Additional element* annex, which is covered at 7.3.10.

7.4.12 Internal code snippets

Internal code snippets can be inserted by using `pre` element.

EXAMPLE

```
<pre>
SCHEMAID = %s"https://www.smpte-ra.org/json-schema/" PUBNUM "/" REVISION [ "/" SHORTNAME]
SHORTNAME = 1*4("/" 1*(ALPHA / DIGIT / "-" / "_" / ".") )
PUBNUM = 1*(DIGIT) ["-" 1*(DIGIT)]
REVISION = STABLEREV
STABLEREV = 4(DIGIT) [2(DIGIT)]
</pre>
```

The `pre` element preserves all white space. As a result, the `pre` element and its contents should not be indented.

7.4.13 Notes

A note is inserted by using a `p` or `div` element whose `class` attribute is equal to `note`. The element element can also have an `id` attribute, in which case the value of the `id` attribute shall start with the string `note-`.

Notes will automatically be formatted and numbered.

EXAMPLE 1

```
<p id="note-example-ref" class="note">
  The license portion of Annex A is verbatim the BSD-3-Clause license
  available at <a>https://spdx.org/licenses/BSD-3-Clause.html</a>.
</p>
```

is rendered as

NOTE — The license portion of Annex A is verbatim the BSD-3-Clause license available at https://spdx.org/licenses/BSD-3-Clause.html.

A note can also be cited by referencing the value of the `id` attribute.

EXAMPLE 2

`<a href="#note-example-ref"></a>` results in 7.4.13, Note.

7.4.14 Examples

An example is inserted by using a `p` or `div` element whose `class` attribute is equal to `example`. The element can also have an `id` attribute, in which case the value of the `id` attribute shall start with the string `ex-`.

Notes will automatically be formatted and numbered.

EXAMPLE 1

```
<div id="ex-example-ref" class="example">The number <i>0.0003</i> can be written
<i>3 × 10−4</i> but is never written <i>3e−4</i>.</div>
```

results in:

EXAMPLE 2

The number 0.0003 can be written 3×10−4 but is never written 3e−4.

An example can also be cited by referencing the value of the `id` attribute.

EXAMPLE 3

`<a href="#ex-example-ref"></a>` results in 7.4.14, Example 2.

7.4.15 Quotes and blockquotes

Prose can be quoted inline using the `q` element or as a block using the `blockquote` element.

EXAMPLE 1

```
<blockquote>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat.</blockquote>
```

results in:

> *Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.*

EXAMPLE 2

```
This is a quote: <q>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
do eiusmod tempor incididunt ut labore et dolore magna aliqua.</q>
```

results in:

This is a quote: *“Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.”*;

7.4.16 Formulae

Formulae are specified using the HTML `math` element.

Unless specified otherwise, a formula is inline.

EXAMPLE 1

```
One needs to evaluate <math><mrow><mi>y</mi><mo>=</mo><mi>a</mi><msup><mi>x</mi><mn>2</mn></msup><mo>+</mo>
<mi>b</mi><mi>x</mi><mo>+</mo><mi>c</mi></mrow></math> to find the position.
```

results in:

One needs to evaluate $y = ax^2 + bx + c$ to find the position.

A block formula is specified by wrapping a single `math` element whose `display` attribute is `"block"` in a `div` element whose class is `formula` and whose `id` attribute starts with `eq-`. Block formulae are numbered and should be referenced: see, for example, Formula (1).

EXAMPLE 2

```
<div class="formula" id="eq-1">
<math display="block"><mrow><mi>y</mi><mo>=</mo><mi>a</mi><msup><mi>x</mi><mn>2</mn></msup><mo>+</mo>
<mi>b</mi><mi>x</mi><mo>+</mo><mi>c</mi></mrow></math>
</div>
```

results in:

$$y = ax^2 + bx + c \tag{1}$$

NOTE — Tools such as https://temml.org/ can be used to convert from Latex to MathML.

The unicode minus sign (`&#x2212;`) is used in math expressions instead of the hyphen-minus, e.g. `&#x2212;42` renders as −42, and `6x &#x2212; 103y` renders as 6x − 103y.

7.4.17 Special Characters

Characters should be encoded as UTF-8-encoded Unicode codepoints, e.g. あ, instead of HTML entities, e.g. `&#x3042;`, except as needed for usage in `pre` or examples.

The characters listed in Table 5 are an exception to this guidance. They are commonly inserted by word processors (e.g. curly quotes substituted for straight quotes, en/em dashes substituted for hyphens) and shall not appear as literal codepoints in the document source. When one of these characters is genuinely needed, it shall be written as a unicode character reference (e.g. `&#x201C;`) so that author intent is explicit.

Table 5 — Characters not permitted as literals in document source

Codepoint	Name	Preview	Reference
U+2018	Left Single Quotation Mark	‘	`&#x2018;`
U+2019	Right Single Quotation Mark	’	`&#x2019;`
U+201C	Left Double Quotation Mark	“	`&#x201C;`
U+201D	Right Double Quotation Mark	”	`&#x201D;`
U+2013	En Dash	–	`&#x2013;`
U+2014	Em Dash	—	`&#x2014;`
U+2011	Non-Breaking Hyphen	-	`&#x2011;`
U+2026	Horizontal Ellipsis	…	`&#x2026;`
U+00A0	No-Break Space	[ ]	` `
U+00AD	Soft Hyphen	[]	`&#xAD;`
U+00A9	Copyright Sign	©	`&#xA9;`
U+00AE	Registered Sign	®	`&#xAE;`

NOTE — The validator reports each literal occurrence with line and column information. Numeric unicode, or named character references (e.g. `&#x201C;`, `&ldquo;`) are recognized as deliberate and are not flagged.

7.4.18 Inline code fragments

Inline code fragments should be wrapped in a `code` element. A code fragment is text that is intended to be used verbatim in computer code, including command line code, including:

- script names (i.e. `smpte.js`)
- element names (i.e. `head`)
- namespaces (i.e. `http://www.smpte-ra.org/schemas/428-7/2014/DCST`)
- short code snippets, e.g. `<meta charset="utf-8" />` or `int i = 3;`
- literal values, e.g., `true` or `00:00:04:ZeroE`
- regular expressions or match pattern, e.g., `YYYY-MM-DD`

NOTE — Namespaces are wrapped with a `code` element and not with an `a` element since they do not necessarily resolve to a resource.

EXAMPLE

```
This is an <code>element</code> that is rendered with the <code>code</code> HTML element.
```

results in:

This is an `element` that is rendered with the `code` HTML element.

7.4.19 Using `class` attributes

In addition to the restriction of using internal or external defined `stylesheet` and `style` elements as per 7.2.15, editors shall not use `style` attributes within `html` elements for styling. Instead, the styling may be modified using `class` values where applicable as listed below.

`annex`

Used on a `section` element to designate the clause as an annex. See 7.3.9.

`note`

Applied to `p`, `div`, or `dd` elements to create a formatted and automatically numbered note. See 7.4.13.

`example`

Applied to `p` or `div` elements to create a formatted and automatically numbered example. See 7.4.14.

`footnote`

Applied to a `p` element inside a `tfoot` element to create a table footnote. Footnotes are only permitted inside a `tfoot` element - use outside of tables is not allowed. See 7.4.10.

`center-cell`;
`right-cell`;
`left-cell`;
`top-cell`

Applied to a `td` element to override the alignment of that individual table cell. See A.1.3.

`text-nowrap`

Applied to an element to prevent line wrapping of its contents. This is commonly used on `th` or `td` elements to keep short identifiers, tags, or codes on a single line. See Clause A.5

`col-1-nowrap` through **`col-8-nowrap`**

Applied to a `table` element to prevent line wrapping within the specified column. See Clause A.6

`col-xsmall`;
`col-small`;
`col-medium`;
`col-wide`

Applied to `th` or `td` elements to suggest a relative column width for that cell. These classes provide extra-narrow, narrow, medium, or wide column layouts without using inline `style` attributes. See Clause A.5

`col-1-xsmall` through **`col-8-xsmall`**;
`col-1-small` through **`col-8-small`**;
`col-1-medium` through **`col-8-medium`**;
`col-1-wide` through **`col-8-wide`**

Applied to a `table` element to suggest relative widths for specific columns. See Clause A.6

`col-1-center` through **`col-8-center`**;
`col-1-right` through **`col-8-right`**;
`col-1-left` through **`col-8-left`**;
`col-1-top` through **`col-8-top`**

Applied to a `table` element to override the alignment of the specified column. Multiple classes may be combined to align multiple columns. See A.1.2.

`list-none`

Applied to a `ul` element to remove bullet markers. See Clause B.4.

`formula`

Applied to a `div` element that contains a block `math` element to format and number displayed formulae. See 7.4.16

`text-right`

Applied to an element to right-align its text content.

`text-left`

Applied to an element to left-align its text content.

`syntax`

Applied to a `td` or `th` element in a syntax listing table to render its contents in a monospace font at a reduced size.

NOTE 1 — All other elements are automatically styled by the default SMPTE defined `CSS` values, such as `"code"` and may not be altered.

NOTE 2 — Tables are automatically formatted for best view on a variety of screen sizes and formats.

8 Tooling

8.1 General

8.1.1 Overview

The SMPTE HTML publication tooling consists of several components:

- Rendering code (`smpte.js`) that renders the author-supplied HTML document into a publication-ready HTML document. The rendering involves numbering clauses, inserting boilerplate, etc., and takes place directly in the web browser used by the author.
- Building code (`scripts/build.mjs`) that automates the process of validating, rendering, generating redlines and uploading publication artifacts to Amazon S3.
- A GitHub workflow (`.github/workflows/main.yml` and `workflows/action.yml`) that automatically executes the building code whenever a change is made to the GitHub repository where the document is tracked.
- Amazon AWS infrastructure where the publication artifacts are deployed.

8.1.2 Rendering

At the heart of the tooling is the rendering code at `smpte.js`.

The rendering code runs in browser and is imported using a `script` element (see 7.2).

8.1.3 Building

Building the document for publication involves the following steps:

1. Validating the document
2. Rendering the document
3. Creating redlines from the rendered document
4. Uploading the rendered document and corresponding redlines to S3

The build process carried by the build script `scripts/build.mjs` is usually executed by the GitHub workflow (see 8.1.4). The script has the following dependencies:

- node dependencies specified in `./package.json`
- HTML 5 validator available as a Python package using `pip install html5validator`

The build process looks for the following environment variables:

- `AWS_S3_REGION`
- `AWS_S3_BUCKET`
- `AWS_S3_KEY_PREFIX`
- `CANONICAL_LINK_PREFIX`

The build process also expects the AWS environment to be set up to allow access to the bucket.

The S3 upload step can be skipped by providing the `validate` argument to the build script.

The build process MAY be configured using an optional `.smpte-build.json` file containing a JSON object that conforms to the JSON schema at Figure 3. When the file is absent, the build derives its configuration from the document metadata and the repository's git tags.

```
{
  "title": "Build configuration",
  "description": "Schema for the SMPTE HTML publication tooling build configuration file",
  "type": "object",
  "properties": {
    "latestEditionTag": {
      "description": "Optional override for the redline base. When set, the build uses this git
commit or tag when generating redlines against the latest edition. When unset or null, the build
auto-derives the redline base from the most recent matching git tag (most recent prior `-pub` tag
when the document's pubStage is PUB; otherwise the most recent prior dated tag of any stage).",
      "type": [ "string", "null" ]
    },
    "publicRepo": {
      "description": "Optional override for the public companion repository (e.g., 'SMPTE/st2098-
2'). When unset, the tooling derives this by stripping the '-private' suffix from the current repo
name. Used by the PCD-push step on release-published events.",
      "type": [ "string", "null" ]
    }
  }
}
```

Figure 3 — Schema for the optional build configuration file `.smpte-build.json`.

If present and not equal to `null`, the `latestEditionTag` property overrides the auto-selected base for the published-edition redline with the specified Git tag or commit. When absent or `null`, the base is auto-selected as described in 8.1.5.

If present and not equal to `null`, the `publicRepo` property overrides the auto-derived public companion repository used by the PCD notification step (see Clause C.7). When absent or `null`, the public repo is derived by stripping the `-private` suffix from the current repository name.

EXAMPLE

```
{
  "latestEditionTag": null
}
```

For the complete description of redline outputs produced by the tooling, see 8.1.5.

8.1.4 GitHub integration

To automatically create and upload to S3 the clean and redline copies of the document when commits are pushed to GitHub, the workflow file at Figure 4 is stored at `.github/workflows/main.yml` in the repository.

```
name: Build SMPTE document

on:
  push:
  pull_request:
  release:
    types: [published]

env:
  AWS_REGION: us-east-1
  AWS_S3_BUCKET: html-doc-pub
  AWS_ROLE: arn:aws:iam::189079736792:role/gh-actions-html-pub
  CANONICAL_LINK_PREFIX: https://doc.smpte-doc.org/

jobs:
  build:
    runs-on: ubuntu-latest
    if: >
      github.repository_owner   == 'SMPTE' && (
      (github.event_name == 'push' && github.ref == 'refs/heads/main')
      || github.event_name == 'pull_request'
      || github.event_name == 'release'
      )
    # These permissions are needed to interact with GitHub's OIDC Token endpoint.
    permissions:
      id-token: write
      contents: write
      pull-requests: write

    steps:

      - name: Checkout repo
        uses: actions/checkout@v3
        with:
          fetch-depth: 0
          submodules: true

      - name: Set repository name
        run: echo "REPOSITORY_NAME=${GITHUB_REPOSITORY#*/}" >> $GITHUB_ENV

      - name: Check out all branches with the exception of the current branch
        run: CUR_BRANCH=$(git rev-parse --abbrev-ref HEAD); for i in `git branch -a | grep remote |
grep -v "remotes/pull" | grep -v HEAD | grep -v ${CUR_BRANCH}`; do git branch --track
${i#remotes/origin/} $i; done
```

```
- name: Configure AWS Credentials
  uses: aws-actions/configure-aws-credentials@v1-node16
  with:
    role-to-assume: ${{ env.AWS_ROLE }}
    aws-region: ${{ env.AWS_REGION }}

- name: Build and deploy document (local)
  uses: ./tooling/workflows
  if: github.repository != 'SMPTE/html-pub'
  with:
    AWS_S3_REGION: ${{env.AWS_REGION}}
    AWS_S3_BUCKET:        ${{env.AWS_S3_BUCKET}}
    AWS_S3_KEY_PREFIX: "${{env.REPOSITORY_NAME}}/"
    CANONICAL_LINK_PREFIX: ${{env.CANONICAL_LINK_PREFIX}}
    GITHUB_TOKEN: ${{secrets.GITHUB_TOKEN}}
    PUBLIC_REPO_PAT: ${{secrets.PUBLIC_REPO_PAT}}

- name: Build and deploy document (HTML Pub repo)
  uses: ./workflows
  if: github.repository == 'SMPTE/html-pub'
  with:
    AWS_S3_REGION: ${{env.AWS_REGION}}
    AWS_S3_BUCKET:        ${{env.AWS_S3_BUCKET}}
    AWS_S3_KEY_PREFIX: "${{env.REPOSITORY_NAME}}/"
    CANONICAL_LINK_PREFIX: ${{env.CANONICAL_LINK_PREFIX}}
    GITHUB_TOKEN: ${{secrets.GITHUB_TOKEN}}
    PUBLIC_REPO_PAT: ${{secrets.PUBLIC_REPO_PAT}}

- name: Notify template repo to update submodule
  if: github.event_name == 'release' && github.repository == 'SMPTE/html-pub'
  env:
    GH_TOKEN: ${{ secrets.TEMPLATE_REPO_PAT }}
    TAG_NAME: ${{ github.event.release.tag_name }}
  run: |
    gh api repos/SMPTE/html-pub-template/dispatches \
      --method POST \
      --field event_type=tooling-release \
      --field "client_payload[tag]=${TAG_NAME}" \
      --field "client_payload[sha]=${GITHUB_SHA}"
```

Figure 4 — GitHub workflow file.

The workflow file defines the following environment variables:

`AWS_S3_REGION`
: Used to set `AWS_S3_REGION`.

`AWS_S3_BUCKET`
: Used to set `AWS_S3_BUCKET`.

`CANONICAL_LINK_PREFIX`
: If set, used to generate the links to publication artifacts that are referenced from the GitHub pull request.

`AWS_ROLE`
: ARN of the AWS role used by the workflow to upload files to S3 (see 8.1.6).

When the workflow is executed on a GitHub pull request, a comment that contains links to the clean and redline documents is automatically added to the pull request. The links use one of the following prefixes:

- `CANONICAL_LINK_PREFIX/AWS_S3_KEY_PREFIX`, if `CANONICAL_LINK_PREFIX` is set;

- `http://AWS_S3_BUCKET.s3-website-AWS_S3_REGION.amazonaws.com/AWS_S3_KEY_PREFIX`, otherwise.

The workflow file automatically sets `AWS_S3_KEY_PREFIX` to the name of the repository.

8.1.5 Redlines

The tooling automatically produces redline (change-tracked) renderings of the document on every build — pull requests, merges to `main`, and release publications. Redlines compare the current rendered document against a chosen reference rendering using the `htmldiff` utility, and the resulting HTML files are deployed to S3 alongside the clean output and bundled into the review zip when applicable.

Three redlines may be produced per build, each with a different reference base:

Redline to current draft (`base-rl.html`)

Generated against the pull request's target branch (typically `main`), via the `GITHUB_BASE_REF` environment variable provided by GitHub Actions on pull request events. Shows what the PR itself changes relative to the branch it will merge into.

Generated: on pull request events. *Omitted:* on push and release events, where no PR base is defined.

Redline to most recent published edition (`pub-rl.html`)

Generated against the most recent prior git tag matching `YYYYMMDD-pub` — the previous published edition. Shows what has changed since the last edition was published. The reference base may be overridden by setting `latestEditionTag` in `.smpte-build.json` — useful when rebuilding an archived edition or pinning to a specific historical release.

This redline satisfies the requirement in SMPTE AG 02 for a redline against the published document (the baseline draft) when preparing a revision.

Generated: on every event. *Omitted:* when no prior `-pub` tag exists (e.g., the first edition).

Redline to most recent release (`release-rl.html`)

Generated against the most recent prior git tag matching `YYYYMMDD-{stage}` for any recognized `pubStage` (`-wd`, `-cd`, `-fcd`, `-dp`, or `-pub`). Shows iterative changes since the last release tag of any kind, supporting draft-to-draft review across the workflow stages defined in SMPTE AG 33.

This redline satisfies the requirement in SMPTE AG 02 that markup from the ballot document be provided for comment resolution.

Generated: when the current document's `pubStage` is `WD`, `CD`, `FCD`, or `DP`. *Omitted:* when `pubStage` is `PUB` (a published edition's relevant comparison is to the prior published edition only) or when no prior dated release tag exists.

In all cases, any tag pointing at the `HEAD` commit is excluded from selection, so a freshly-tagged release does not redline against itself. When the build runs on a pull request, links to all available redline outputs are included in the comment posted to the PR (see 8.1.4). When the build runs on a release publication, the redline files are bundled into the review zip attached to the release (see Clause C.6).

8.1.6 Amazon AWS integration

The tooling uploads publication artifacts to an S3 bucket according to the following parameters:

`AWS_S3_REGION`

Region where the S3 bucket lives, e.g., `us-west-1`

`AWS_S3_BUCKET`

Name of the S3 bucket, e.g., `html-doc-pub`

`AWS_S3_KEY_PREFIX`

Prefix used when creating the S3 key where the document is stored, e.g., `draft/ag05/`

EXAMPLE

Given `AWS_S3_BUCKET=html-doc-pub` and `AWS_S3_KEY_PREFIX=ag-05/`, the tooling uploads publication artifacts under the prefix `s3://html-doc-pub/ag05/`.

An OIDC role grants the tooling permissions to upload objects to the bucket. This avoids maintaining IAM users and storing GitHub secrets.

The trust policy for the role associated with one GitHub repository is specified at Figure 5.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::189079736792:oidc-
provider/token.actions.githubusercontent.com"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
                },
                "StringLike": {
                    "token.actions.githubusercontent.com:sub": "repo:SMPTE/*"
                }
            }
        }
    ]
}
```

Figure 5 — Trust policy for the OIDC role.

The permissions for the role associated with one GitHub repository is specified at Figure 6.

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": "s3:PutObject",
          "Resource": "arn:aws:s3:::<name of bucket>/*"
      }
  ]
}
```

Figure 6 — Permissions granted to the role associated with each GitHub repository, where `<name of bucket>` is replaced by `AWS_S3_BUCKET`.

The bucket is made public using the policy at Figure 7.

```
{
  "Version": "2008-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Principal": "*",
          "Action": "s3:GetObject",
          "Resource": "arn:aws:s3:::<name of bucket>/*"
      }
  ]
}
```

Figure 7 — Public bucket policy where `<name of bucket>` is replaced by `AWS_S3_BUCKET`

The contents of the bucket are also made available through a Cloudfront distribution, which allows the publication artifacts to be made available through a custom domain name and TLS (currently `https://doc.smpte-doc.org/`).

8.1.7 Git commit hooks

It is advisable to install the git commit hook listed at Figure 8. It detects errors before a document can be committed.

```
#!/bin/sh
if test -f "tooling/scripts/validate.mjs"; then
  validate=./tooling/scripts/validate.mjs
else
  validate=./scripts/validate.mjs
fi

node $validate ./doc/main.html
```

Figure 8 — Pre-commit hook that validates the main element.

8.2 Using tooling locally

8.2.1 Overview

Many of the features and automatic formatting available in the tooling is present and available when working locally. This allows an editor to create and edit a valid HTML document before pushing to a repo and encountering an error.

It is recommended to utilize VS Code (https://code.visualstudio.com/) as your local HTML editor, using the Live Preview extension (https://marketplace.visualstudio.com/items?itemName=ms-vscode.live-server), which shows some validation in real-time for an editor.

Other useful extension(s) for VS Code are:

- Git Graph (visual aid) - https://marketplace.visualstudio.com/items?itemName=mhutchie.git-graph

8.2.2 Installation and prerequisites

To use the full features locally either on command line, or in termimal (also available inside of VS Code), an editor will need to have the following installed locally:

- Node.js - https://nodejs.org/en/download

- Python - https://www.python.org/downloads/
- PIP - https://pip.pypa.io/en/stable/installation/

Once installed, it is recommended to keep up to date with the latest versions.

8.2.3 Repo setup

Assuming the prerequisites listed in 8.2.2 are already installed, each time a new repo is cloned and work is to be started, the repo will need to be configured for its dependencies, based on the current `./package.json` file.

The command for this is:

```
npm install
```

NOTE — This command will update `./package-lock.json`, and can be run again if changes are made to `./package.json` for editing and/or updating dependencies based on the repo.

Once npm/Node is set up, it is also recommended to set up the requirements and further dependencies for validation to run properly.

The command for this is:

```
npm run prevalidate
```

This will check and (if needed) install the HTML 5 validator as noted in 8.1.3. If the HTML 5 validator has already been installed, or the command has been run before (on another repo), the prompt will return:

```
html5validator found. HTML validation ready.
```

8.2.4 Updating the tooling submodule

Document repositories include the tooling (`./tooling/`) as a git submodule. When the tooling is updated, each document repo must be manually bumped to point to the new tooling commit. It is important to keep the submodule as up to date as possible, as updates may include bug fixes, validation improvements, and formatting changes that affect document output.

NOTE 1 — Individual changes to files within the `./tooling/` submodule directory are not permitted. All tooling changes must be made through the https://github.com/SMPTE/html-pub repository and picked up via a submodule bump.

To check which commit the submodule is currently pinned to and whether it is behind the remote, run:

```
git submodule status
```

The output lists each submodule with its current commit SHA. To determine whether the submodule is current, compare the commit SHA against the latest release tag and commit at https://github.com/SMPTE/html-pub.

To update the submodule to the latest commit on its tracked remote branch, run:

```
git submodule update --remote
```

NOTE 2 — This command fetches from the submodule's remote and advances the pointer in the superproject to the latest available commit on the tracked branch. It does not automatically stage or commit the change in the parent repo.

After updating, the parent repository will show the submodule directory as modified. Stage and commit the bump with:

```
git add tooling/
git commit -m "Bump tooling to latest"
```

Push the change to the remote as normal:

```
git push
```

8.2.5 Validation

When creating a PR on GitHub, the tooling automatically performs a build process as noted in 8.1.3, and part of that process is doing validation, both the internal to the tooling and for general HTML 5 validation.

While the internal tooling validation occurs at runtime while editing, waiting until the build action during a PR to check the HTML 5 validation can present an editor with unforseen issues (i.e. an invalid element or broken schema error), which then results in failures on the PR/build that must be fixed. Often times when this occurs the preview with redlines is not available in the PR, and/or the tooling will not allow a build to finish. Therefore it is highly recommended an editor runs the validation locally prior to pushing to GitHub to prevent these failures frmo occuring.

The command for this is:

```
npm run validate
```

NOTE — This command reports back errors in (2) seperate sections that are required to be resolved before a build can complete. The HTML validation includes line numbers for the convenience of the editor.

EXAMPLE

```
> smpte-html-pub-tooling@2.1.0 prevalidate
> node scripts/ensure-html5validator.mjs

html5validator found. HTML validation ready.

> smpte-html-pub-tooling@2.1.0 validate
> run-p validate:doc validate:html

> smpte-html-pub-tooling@2.1.0 validate:html
> html5validator --errors-only ./doc/main.html

> smpte-html-pub-tooling@2.1.0 validate:doc
> node ./scripts/validate.mjs ./doc/main.html

Unknown element in clause HTMLElement {
  [Symbol(SameObject caches)]: [Object: null prototype] { children: HTMLCollection {} }
}
".../SMPTE/html-pub/./doc/main.html":1512.7-1512.10: error: No "p" element in scope but a "p" end tag seen.
".../SMPTE/html-pub/./doc/main.html":1524.7-1524.10: error: No "p" element in scope but a "p" end tag seen.
".../SMPTE/html-pub/./doc/main.html":1554.21-1555.7: error: Saw "<" when expecting an attribute name. Probable
cause: Missing ">" immediately before.
".../SMPTE/html-pub/./doc/main.html":1554.21-1555.9: error: End tag had attributes.
".../SMPTE/html-pub/./doc/main.html":1555.242-1555.245: error: No "p" element in scope but a "p" end tag seen.
ERROR: "validate:html" exited with 5.
```

8.2.6 Handling new validation errors after a tooling bump

A tooling update may introduce new or stricter validation rules. After bumping the submodule, run validation locally as described in 8.2.5 before committing or pushing.

If validation reports new errors that were not present before the bump, those errors must be resolved in the document source before the bump commit is pushed. Do not push a submodule bump that causes a document to fail validation.

NOTE — Validation errors introduced by a tooling update reflect real issues in the document source. They should be corrected in the document, not worked around by reverting or pinning to an older tooling commit.

Annex A
Table Examples (Informative)

A.1 Alignment

A.1.1 Default Alignment

By default, table headers within `thead` are set to `text-align: center;`, and cells within the body `tbody` are set to `text-align: left;`, as the below example shows.

```
<table id="tab-sample-tabledata-defaultalign">
  <caption>Default Alignment Table Sample</caption>
  <thead>
    <tr>
      <th>Sample Number</th>
      <th>Sample Name</th>
      <th>Sample Code</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>0001</td>
      <td>Name 01</td>
      <td><code>Code 1</code></td>
    </tr>
    <tr>
      <td>0002</td>
      <td>Name 02</td>
      <td><code>Code 2</code></td>
    </tr>
    <tr>
      <td>0003</td>
      <td>Name 03</td>
      <td><code>Code 3</code></td>
    </tr>
  </tbody>
</table>
```

is rendered as

Table A.1 — Default Alignment Table Sample

Sample Number	Sample Name	Sample Code
0001	Name 01	`Code 1`
0002	Name 02	`Code 2`
0003	Name 03	`Code 3`

A.1.2 Column Alignment

To change the alignment of an entire column, classes such as `col-3-center`, `col-3-right`, `col-3-left`, or `col-3-top` may be added to the `table` element. Multiple alignment classes may be combined across different columns, and multiple classes targeting the same column are also allowed since horizontal (`text-align`) and vertical (`vertical-align`) alignment are independent properties, as the below example shows.

```
<table id="tab-sample-tabledata-colcenter" class="col-1-top col-2-top col-2-right col-3-top col-3-center">
  <caption><code>col-1-top col-2-top col-2-right col-3-top col-3-center</code> Table Sample</caption>
  <thead>
    <tr>
      <th>Sample Number</th>
      <th>Value</th>
      <th>Sample Code</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>0001</td>
      <td>42</td>
      <td><code>Code 1</code></td>
    </tr>
    <tr>
      <td>0002</td>
      <td>This is a longer description that spans multiple lines to demonstrate how <code>col-2-top</code> and
<code>col-2-right</code> may be combined on the same column to independently control vertical and horizontal
alignment.</td>
      <td><code>Code 2</code></td>
    </tr>
    <tr>
      <td>0003</td>
      <td>8</td>
      <td><code>Code 3</code></td>
    </tr>
  </tbody>
</table>
```

is rendered as

Table A.2 — `col-1-top col-2-top col-2-right col-3-top col-3-center` Table Sample

Sample Number	Value	Sample Code
0001	42	`Code 1`
0002	This is a longer description that spans multiple lines to demonstrate how `col-2-top` and `col-2-right` may be combined on the same column to independently control vertical and horizontal alignment.	`Code 2`
0003	8	`Code 3`

NOTE 1 — Alignment classes are available for columns 1-8 using `col-x-center`, `col-x-right`, `col-x-left`, and `col-x-top`. Multiple classes may be combined on the same `table` to align different columns independently.

NOTE 2 — Tables over 8 columns wide should not be used and instead split into multiple smaller tables.

A.1.3 Cell Alignment

To change the alignment of an individual cell, classes such as `center-cell`, `right-cell`, `left-cell`, or `top-cell` may be added to each `td` as needed, as the below example shows.

```
<table id="tab-sample-tabledata-centercell">
  <caption>Cell Alignment Table Sample</caption>
  <thead>
    <tr>
      <th>Sample Number</th>
      <th>Sample Name</th>
      <th>Sample Code</th>
```

```
    </tr>
  </thead>
  <tbody>
    <tr>
      <td class="right-cell">0001</td>
      <td>Name 01</td>
      <td><code>Code 1</code></td>
    </tr>
    <tr>
      <td class="center-cell">0002</td>
      <td>Name 02</td>
      <td><code>Code 2</code></td>
    </tr>
    <tr>
      <td class="top-cell">0003</td>
      <td>Name 03</td>
      <td><code>Code 3</code></td>
    </tr>
    <tr>
      <td class="top-cell">0004</td>
      <td class="top-cell">Name 04</td>
      <td>This is a longer description that spans multiple lines to demonstrate how the <code>top-cell</code>
class aligns the content of the first two cells to the top of the row rather than the default middle alignment
when adjacent cells contain more content.</td>
    </tr>
  </tbody>
</table>
```

is rendered as

Table A.3 — Cell Alignment Table Sample

Sample Number	Sample Name	Sample Code
0001	Name 01	`Code 1`
0002	Name 02	`Code 2`
0003	Name 03	`Code 3`
0004	Name 04	This is a longer description that spans multiple lines to demonstrate how the `top-cell` class aligns the content of the first two cells to the top of the row rather than the default middle alignment when adjacent cells contain more content.

NOTE — Cell alignment classes may be added to as many `td` elements as needed and may be combined with column-level alignment classes.

A.2 Multiple Headers

To insert additional header(s), use `th` in place of `td` within the `tr` of `tbody` that will contain the additional header, as the below example shows.

EXAMPLE

```
<table id="tab-sample-tabledata-multiheader">
  <caption>Multiple Header Table Sample</caption>
  <thead>
    <tr>
      <th>Sample Number</th>
      <th>Sample Name</th>
      <th>Sample Code</th>
    </tr>
```

```
  </thead>
  <tbody>
    <tr>
      <td>0001</td>
      <td>Name 01</td>
      <td><code>Code 1</code></td>
    </tr>
    <tr>
      <td>0002</td>
      <td>Name 02</td>
      <td><code>Code 2</code></td>
    </tr>
    <tr>
      <th>Sample Number 2</th>
      <th>Sample Name 2</th>
      <th>Sample Code 2</th>
    </tr>
    <tr>
      <td>0003</td>
      <td>Name 03</td>
      <td><code>Code 3</code></td>
    </tr>
  </tbody>
</table>
```

is rendered as

Table A.4 — Multiple Header Table Sample

Sample Number	Sample Name	Sample Code
0001	Name 01	`Code 1`
0002	Name 02	`Code 2`
Sample Number 2	**Sample Name 2**	**Sample Code 2**
0003	Name 03	`Code 3`

A.3 Cell Spanning

A.3.1 `colspan`

`colspan` is supported as needed, as the below example shows.

```
<table id="tab-sample-tabledata-colspan">
  <caption><code>colspan</code> Table Sample</caption>
  <thead>
    <tr>
      <th>Sample Number</th>
      <th>Sample Name</th>
      <th>Sample Code</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>0001</td>
      <td>Name 01</td>
      <td><code>Code 1</code></td>
    </tr>
    <tr>
      <td>0002</td>
```

```
      <td>Name 02</td>
      <td><code>Code 2</code></td>
    </tr>
    <tr>
      <td colspan="3">next section</td>
    </tr>
    <tr>
      <td>0003</td>
      <td>Name 03</td>
      <td><code>Code 3</code></td>
    </tr>
  </tbody>
</table>
```

is rendered as

Table A.5 — `colspan` Table Sample

Sample Number	Sample Name	Sample Code
0001	Name 01	`Code 1`
0002	Name 02	`Code 2`
next section		
0003	Name 03	`Code 3`

NOTE — See https://html.com/tables/rowspan-colspan/ for additional information.

A.3.2 `rowspan`

`rowspan` is supported as needed, as the below example shows.

```
<table id="tab-sample-tabledata-rowspan">
  <caption><code>rowspan</code> Table Sample</caption>
  <thead>
    <tr>
      <th>Sample Number</th>
      <th>Sample Name</th>
      <th>Sample Code</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td rowspan="2">0001</td>
      <td>Name 01</td>
      <td><code>Code 1</code></td>
    </tr>
    <tr>
      <td>Name 02</td>
      <td><code>Code 2</code></td>
    </tr>
    <tr>
      <td>0002</td>
      <td>Name 03</td>
      <td><code>Code 3</code></td>
    </tr>
  </tbody>
</table>
```

is rendered as

Table A.6 — `rowspan` Table Sample

Sample Number	Sample Name	Sample Code
0001	Name 01	`Code 1`
	Name 02	`Code 2`
0002	Name 03	`Code 3`

NOTE — See https://html.com/tables/rowspan-colspan/ for additional information.

A.4 Complex Example

All the various options above are available to be combined as needed to create complex tables, as the below example shows.

```
<table id="tab-sample-tabledata-complex" class="col-3-center col-4-center">
  <caption>Complex Table Sample</caption>
  <thead>
    <tr>
      <th>Sample Number</th>
      <th>Sample Name</th>
      <th>Sample Code</th>
      <th>Sample Data</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td rowspan="2">0001</td>
      <td>Name 01</td>
      <td><code>Code 1</code></td>
      <td>Data 01</td>
    </tr>
    <tr>
      <td>Name 02</td>
      <td class="center-cell"><code>Code 2</code></td>
      <td>Data 01</td>
    </tr>
    <tr>
      <td>0003</td>
      <td>Name 03</td>
      <td><code>Code 3</code></td>
      <td>Data 01</td>
    </tr>
    <tr>
      <th>Sample Number 2</th>
      <th>Sample Name 2</th>
      <th>Sample Code 2</th>
      <th>Sample Data 2</th>
    </tr>
    <tr>
      <td>0004</td>
      <td>Name 04</td>
      <td><code>Code 5</code></td>
      <td>Data 05</td>
    </tr>
    <tr>
      <td colspan="3" class="center-cell">section of data</td>
      <td class="center-cell">Data 05.5</td>
    </tr>
    <tr>
      <td>0005</td>
      <td>Name 05</td>
```

```
      <td><code>Code 5</code></td>
      <td>Data 06</td>
    </tr>
  </tbody>
</table>
```

is rendered as

Table A.7 — Complex Table Sample

Sample Number	Sample Name	Sample Code	Sample Data
0001	Name 01	`Code 1`	Data 01
	Name 02	`Code 2`	Data 01
0003	Name 03	`Code 3`	Data 01
Sample Number 2	**Sample Name 2**	**Sample Code 2**	**Sample Data 2**
0004	Name 04	`Code 5`	Data 05
section of data			Data 05.5
0005	Name 05	`Code 5`	Data 06

A.5 Column Width and No-wrap (Cell-level)

Column widths and non-wrapping text may be controlled at the individual cell level using classes such as `col-small`, `col-wide`, and `text-nowrap`.

EXAMPLE

```
<table id="tab-sample-tabledata-widthcell">
  <caption>Cell-level Width and No-wrap Table Sample</caption>
  <thead>
    <tr>
      <th class="col-small text-nowrap">ID</th>
      <th>Description</th>
      <th class="col-wide">Identifier</th>
      <th class="text-nowrap">Code</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td class="col-small text-nowrap">01</td>
      <td>Example element</td>
      <td class="col-wide">urn:smpte:example:identifier:0001</td>
      <td class="text-nowrap">A-01</td>
    </tr>
    <tr>
      <td class="col-small text-nowrap">02</td>
      <td>Another example element</td>
      <td class="col-wide">urn:smpte:example:identifier:0002</td>
      <td class="text-nowrap">B-02</td>
    </tr>
  </tbody>
</table>
```

is rendered as

Table A.8 — Cell-level Width and No-wrap Table Sample

ID	Description	Identifier	Code
01	Example element	urn:smpte:example:identifier:0001	A-01
02	Another example element	urn:smpte:example:identifier:0002	B-02

A.6 Column Width and No-wrap (Column-level)

Column width and wrapping behavior may also be controlled for entire columns using table-level classes such as `col-3-nowrap` and column width classes.

EXAMPLE

```
<table id="tab-sample-tabledata-widthcolumn" class="col-3-nowrap col-1-xsmall col-4-wide">
  <caption>Column-level Width and No-wrap Table Sample</caption>
  <thead>
    <tr>
      <th>ID</th>
      <th>Name</th>
      <th>Code</th>
      <th>Identifier</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>01</td>
      <td>Element One</td>
      <td>ABC-0001</td>
      <td>urn:smpte:example:identifier:0001</td>
    </tr>
    <tr>
      <td>02</td>
      <td>Element Two</td>
      <td>ABC-0002</td>
      <td>urn:smpte:example:identifier:0002</td>
    </tr>
  </tbody>
</table>
```

is rendered as

Table A.9 — Column-level Width and No-wrap Table Sample

ID	Name	Code	Identifier
01	Element One	ABC-0001	urn:smpte:example:identifier:0001
02	Element Two	ABC-0002	urn:smpte:example:identifier:0002

Annex B
List Examples (Informative)

B.1 Customizing Ordered Lists

B.1.1 General

Ordered lists (`ol`) may be modified from their default list numberings (numbers) using the `type` attribute. The possible list numberings are lowercase letter (`a`), uppercase letter (`A`), lowercase Roman numerals (`i`), uppercase Roman numerals (`I`), and numbers (`1` - default if not used). The `start` attribute may also be used with a numeric value to denote the start of the numbering of the listed items.

NOTE — The `type` and `start` attributes are not supported for unordered lists (`ul`). Formatting for bullets on `ul` lists are auto-generated by the tooling, based on nesting.

B.1.2 List `type` Examples

See below for various list examples utilizing `type` and `start`

EXAMPLE 1

Lower Case Letter:

```
<ol type="a">
  <li>First list item</li>
  <li>Second list item</li>
</ol>
```

is rendered as:

a. First list item

b. Second list item

EXAMPLE 2

Upper Case Letter:

```
<ol type="A">
  <li>First list item</li>
  <li>Second list item</li>
</ol>
```

is rendered as:

A. First list item

B. Second list item

EXAMPLE 3

Lower Case Roman Numeral:

```
<ol type="i">
  <li>First list item</li>
  <li>Second list item</li>
</ol>
```

is rendered as:

i. First list item

ii. Second list item

EXAMPLE 4

Upper Case Roman Numeral:

```
<ol type="I">
  <li>First list item</li>
  <li>Second list item</li>
</ol>
```

is rendered as:

I. First list item

II. Second list item

EXAMPLE 5

List starting from (3):

```
<ol start="3">
  <li>First list item</li>
  <li>Second list item</li>
</ol>
```

is rendered as:

3. First list item

4. Second list item

B.2 List Nesting

Both `ol` and `ul` elements may be nested as many times within a `li` element as needed to achieve the required results. See below for various examples.

EXAMPLE 1

Nesting `ol`:

```
<ol>
  <li>First list item
    <ol type="a">
      <li>First sub item</li>
      <li>Second sub item</li>
    </ol>
  </li>
  <li>Second list item</li>
</ol>
```

is rendered as:

1. First list item

    a. First sub item

    b. Second sub item

2. Second list item

NOTE — When nesting `ol` style lists, different `type` attributes should be used to ensure readablity, as this is not achieved automatically.

EXAMPLE 2

Nesting `ul`:

```
<ul>
  <li>First list item
    <ul>
      <li>First sub item</li>
      <li>Second sub item</li>
    </ul>
  </li>
  <li>Second list item</li>
</ul>
```

is rendered as:

- First list item
    - First sub item
    - Second sub item
- Second list item

EXAMPLE 3

Nesting `ul` in `ol`:

```
  <ol>
  <li>First list item
    <ul>
      <li>First sub item</li>
      <li>Second sub item</li>
    </ul>
  </li>
  <li>Second list item</li>
  </ol>
```

is rendered as:

1. First list item
    - First sub item
    - Second sub item
2. Second list item

B.3 Complex List Example

The features in Clause B.1 and Clause B.2 may be combined as needed to achieve the required results for a complex list. See below for an example.

EXAMPLE

```
<ol>
  <li>First list item
    <ol type="a">
```

```
      <li>First sub item
        <ol type="i">
          <li>First sub item's 1st item with small roman numerals</li>
          <li>First sub item's 2nd item with small roman numerals</li>
        </ol>
      </li>
      <li>Second sub item
        <ol type="i" start="3">
          <li>Second sub item's 1st item with small roman numerals, starting with "iii" to continue
numbers</li>
          <li>Second sub item's 2nd item with small roman numerals</li>
        </ol>
      </li>
    </ol>
  </li>
  <li>Second list item</li>
  <li>Third list item</li>
  <li>Forth list item
    <ol type="a">
      <li>Forth list item's sub item
        <ul>
          <li>1st bullet under sub item
            <ul>
              <li>1st sub bullet</li>
            </ul>
          </li>
          <li>2nd bullet under sub item</li>
        </ul>
      </li>
      <li>Forth list item's sub item</li>
    </ol>
  </li>
  <li>Fifth list item
    <ul>
      <li>1st bullet sub item</li>
      <li>2nd bullet sub item</li>
    </ul>
  </li>
</ol>
```

is rendered as:

1. First list item
    a. First sub item
        i. First sub item's 1st item with small roman numerals
        ii. First sub item's 2nd item with small roman numerals
    b. Second sub item
        iii. Second sub item's 1st item with small roman numerals, starting with "iii" to continue numbers
        iv. Second sub item's 2nd item with small roman numerals
2. Second list item
3. Third list item
4. Forth list item
    a. Forth list item's sub item
        - 1st bullet under sub item
            - 1st sub bullet

        - 2nd bullet under sub item

    b. Forth list item's sub item

5. Fifth list item

    - 1st bullet sub item
    - 2nd bullet sub item

B.4 Unordered List Without Bullets

To remove bullet markers from an unordered list, the class `list-none` may be applied to the `ul` element, as the below example shows.

EXAMPLE

```
<ul class="list-none">
  <li>First list item</li>
  <li>Second list item</li>
  <li>Third list item</li>
</ul>
```

is rendered as:

First list item

Second list item

Third list item

Annex C
Auto-draft releases (Normative)

C.1 General

This annex describes the tooling behavior that auto-drafts GitHub releases. For the document workflow that drives *when* releases are created and *who* publishes them, see SMPTE AG 33.

C.2 Trigger and auto-draft

Whenever the state of the document changes to `<meta itemprop="pubState" content="pub" />` on the `main` branch, the tooling automatically creates a *draft* release in the GitHub repo. A user with appropriate permissions reviews and publishes the draft to finalize the release and trigger production of the associated artifacts.

C.3 Tag and release naming

The GitHub release and the git tag shall be named `YYYYMMDD "-" pubStage`, where `YYYYMMDD` represents the value of `pubDateTime` (as defined in 7.2.12) and the `pubStage` suffix is the value of the `pubStage` meta in lowercase.

EXAMPLE
`20230615-fcd` or `20230925-pub`

C.4 Same-day iterations

For workflow events that share both `pubDateTime` and `pubStage` with the most recent release but represent a distinct release moment, the merge commit message shall include the token `[force-release]`. The tooling appends a numeric suffix (`-2`, `-3`, …) to the tag to disambiguate.

C.5 Publishing the draft release

To publish a draft release after it has been auto-created:

1. In the repository on GitHub, navigate to the **Releases** page (`https://github.com/SMPTE/{repo}/releases`).
2. Open the draft release whose name matches the expected `YYYYMMDD-pubStage` tag for the merged document state.
3. Click **Edit**, review the auto-generated release notes, and confirm that the target commit corresponds to the merge that produced the new state.
4. Click **Publish release**.

C.6 Release artifacts

Publishing the draft release attaches zip packages to the release details page:

- **Ballot zip** — named per SMPTE AG 02. Bundles the rendered document together with the published-edition redline (`pub-rl.html`) and the most-recent-release redline (`release-rl.html`). See 8.1.5 for redline details and reference-base selection.
- **Publish zip** — named by the tag, used for publishing per SMPTE AG 33.

EXAMPLE

See https://github.com/SMPTE/st428-24-private/releases/tag/20241101-dp, where https://github.com/SMPTE/st428-24-private/releases/download/20241101-dp/27c-st-428-24-dp-2024-12-06-pub.zip (ballot zip) and https://github.com/SMPTE/st428-24-private/releases/download/20241101-dp/20241101-dp.zip (publish zip) are available.

C.7 Public Committee Draft (PCD) notification of the public repo

C.7.1 Trigger and public-repo resolution

When the document state on a published release is `pubState=pub`, `pubStage=CD`, and `pubConfidential=no` — uniquely identifying a Public Committee Draft per SMPTE AG 33 — the tooling additionally notifies the public companion repository so that its `README.md` is updated with a pointer to the rendered document and the public-review comment-period info.

The public repository is resolved as follows:

1. If `.smpte-build.json` sets a non-empty `publicRepo` property (see Figure 3), that value is used verbatim (e.g., `SMPTE/st2098-2`).
2. Otherwise, the private repository name is taken from `GITHUB_REPOSITORY` and its `-private` suffix is stripped (e.g., `SMPTE/st2098-2-private` → `SMPTE/st2098-2`).
3. If no public repo can be resolved (the private repo name does not end in `-private` and no override is set), the PCD notification step is skipped.

C.7.2 Auto-populated draft release template

When the document state at the time of draft creation is in the PCD combination (`pubState=pub`, `pubStage=CD`, `pubConfidential=no`), the tooling automatically appends a fill-in template to the draft release body. The template is added below the auto-generated changelog, separated by a horizontal rule, in the following shape:

```
## PCD review period

End 1: YYYY-MM-DD
End 2: YYYY-MM-DD

Message: Enter marketing message. If a revision, also describe what changed in this revision.
```

The template is appended only when the draft is created for a PCD-eligible state. Releases for other stages (e.g., `FCD`, `DP`, or final `PUB`) do not receive the template, and their release notes contain only the auto-generated changelog.

To prepare the release for publication, the editor opens the draft in the GitHub **Releases** page, replaces the two `YYYY-MM-DD` placeholders with the actual review-period dates, and replaces the placeholder message text with the marketing description for the PCD. The keyword lines (`End 1:`, `End 2:`, `Message:`) shall remain at the start of their respective lines so that the tooling can parse them after publish.

`End 1` is the earliest date on which the public review period may end; `End 2` is the latest. These map directly to the SMPTE phrasing *"no earlier than X, and no later than Y"* in the public README. The `Message:` value may span multiple paragraphs to the end of the release body, and is rendered verbatim as the body of the *Details* section in the public README.

C.7.3 Release-notes preservation on publish

On publish, the tooling assembles the final release body by combining the publication links generated by the build (clean rendering, redlines, ZIP packages) with the existing body content (the auto-generated changelog and any PCD keywords filled in by the editor). The combined body has the publication links at the top, a horizontal-rule separator, then the existing content

below. None of the editor-supplied content is discarded on publish; the publication links are added alongside, not in place of, the changelog and PCD information.

C.7.4 Cross-repo dispatch

The tooling fires a `repository_dispatch` event of type `pcd-published` on the public repository with a payload containing the tag, release URL, document URL (constructed from `CANONICAL_LINK_PREFIX` + private repo name + tag), the two end dates, the message, and identifiers for both repositories. The public repository's workflow listens for this event and opens a pull request updating its `README.md` — specifically, the block bounded by `<!-- PCD-INFO:START -->` and `<!-- PCD-INFO:END -->` markers between the `_This repository is public._` anchor line and the `## Notices` heading.

The PCD notification step is gated by the availability of the `PUBLIC_REPO_PAT` input on the build action, which carries write access to `repository_dispatch` on the public repo. When the input is unset, the notification is silently skipped (the rest of the release-published flow runs unchanged).

The step is idempotent across re-runs: before dispatching, the tooling checks whether the public repository already has a `pcd-info/{tag}` branch. If present, the notification is skipped — the pull request has already been opened (or merged) by a prior run.

C.7.5 Public-repository setup

Public repositories opt in to receiving PCD notifications by installing a thin workflow file (`.github/workflows/pcd-update.yml`) that listens for `repository_dispatch` events of type `pcd-published` and invokes the reusable workflow `SMPTE/html-pub/.github/workflows/pcd-update-reusable.yml` at the appropriate tooling tag. See the reusable workflow for the supported inputs.

C.7.6 End-of-period cleanup

After the public review period ends, the public repository's `README.md` should no longer advertise an expired PCD. The cleanup is handled entirely on the public side by a scheduled workflow that runs once per day, parses the *"no later than"* date from its own README, and opens a pull request to clear the PCD-INFO block when that date has passed.

Public repositories opt in by installing a second thin workflow file (`.github/workflows/pcd-cleanup.yml`) alongside the PCD-update wrapper. The cleanup workflow triggers on a daily cron schedule (`0 6 * * *` — 06:00 UTC) and on manual `workflow_dispatch`. The dispatch surfaces an optional **force** input that bypasses the date check — useful if a PCD is cancelled before its scheduled end, or for testing.

The cleanup logic:

1. Read the public repository's `README.md`.
2. If the file lacks the `<!-- PCD-INFO:START -->` markers, exit (nothing to clear).
3. Extract the `no later than YYYY-MM-DD` date from inside the marker block.
4. Compare to today's UTC date. If today is after that date (or the `force` dispatch input is set), proceed; otherwise exit.
5. Invoke a reusable workflow on the tooling that rewrites the PCD-INFO block, replacing its content with collapsed adjacent markers (`<!-- PCD-INFO:START --><!-- PCD-INFO:END -->`).
6. Open a pull request on a stable branch named `pcd-info-cleanup` with the cleared README.

The collapsed markers remain in the README as a permanent insertion point. Any subsequent PCD round refills them via the standard dispatch flow (see C.7.4) without re-inserting the markered block. The cleanup is idempotent across daily runs: once

the block is cleared, subsequent runs detect the already-collapsed markers and no new pull request is created.

Annex D
Additional elements (Informative)

The following are the non-prose elements of this document:

a. Sample text element (informative). file: <sample.txt>.

b. Sample URL element (normative). url: <https://github.com/SMPTE/html-pub/tree/631ed7a0312fa183c356a7b742129ce46f92736f>.

Bibliography

ISO/IEC Directives, Part 2, Principles and rules for the structure and drafting of ISO and IEC documents (Ninth edition, 2021)
url: https://www.iso.org/sites/directives/current/part2/index.xhtml

SMPTE AG 16, Standards Style Guide
url: https://doc.smpte-doc.org/ag-16/main/