PDF association

Specification

Deriving HTML from PDF

1.0
2019-06

Deriving HTML
from PDF TWG

PDF Association
Friedenstr. 2A · 16321 Bernau bei Berlin · Germany

E-mail: copyright@pdfa.org
Web: pdfa.org
Published in Germany and the United States of America

Foreword

The PDF Association is the meeting place of the PDF industry. The work of preparing industry standards and best practices is normally carried out through Technical Working Groups (TWGs). The results of such work may, if desired by the members of the respective TWG, the Board of Directors, and the members as a whole, may be submitted to ISO for publication as an International Standard.

Each PDF Association member interested in a subject for which a TWG has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with the PDF Association, also take part in the work. The PDF Association collaborates closely with the 3D PDF Consortium and ISO on all matters of standardization.

The procedures used to develop this document and those intended for its maintenance are described in the PDF Association's publication process.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The PDF Association shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

Table of Contents

Introduction

Over the past 25 years PDF format has matured from a fixed-layout, page-description format into a sophisticated foundation for deploying content. In 2018, PDF's dominance in the electronic document marketplace remains based on its fixed-layout heritage rather than its capabilities as a rich content container.

In the modern world of small devices, IoT and connected systems, where interchange and reuse of data is critical, it is reasonable to question the continued relevance of PDF's core value proposition. In particular, search engines, machine learning and artificial intelligence systems focus on accessing information contained in documents over visual representation. In other cases, document producers wish to deliver data in a form that is suitable for automated processing while using a PDF file as a record for trust purposes. End users want electronic documents that adapt smoothly to viewing on diverse small devices.

By describing the algorithm that produces conforming HTML from a tagged PDF, this document shows how well-tagged PDF documents, containing both traditional fixed-layout content and the semantic structures leveraged by modern devices and software, can be reliably and consistently reused as HTML to support better user experiences and renew PDF's value proposition.

HTML was chosen as a derivation target because HTML is consumed on all platforms and supported by all major vendors. With small modifications, developers can use this document to export content from well-tagged PDF to any format.

Author

Roman Toda, Normex

Contributors

Boris Doubrov, Dual Lab

Olaf Drümmer, callas software

Matthew Hardy, Adobe

Duff Johnson, PDF Association

Leonard Rosenthol, Adobe

References

ISO 14289-1:2014, Document management — Electronic Document File Format Enhancement for Accessibility — Part 1: Use of ISO 32000-1 (PDF/UA-1)

ISO/IEC 16262:2011, Information technology — Programming languages, their environments and system software interfaces — ECMAScript language specification. (Also known as JavaScript. Also available as ECMA-262 Edition 5.1 from ECMA)

ISO 21757-1, Document management – ECMAScript for PDF – Part 1: Use of ISO 32000-2 (PDF 2.0)

NOTE 1 As of this writing, this document is at ISO's Committee Draft stage, and is available only to accredited members of ISO TC 171 SC 2 WG 8, or to members of the PDF Association.

ISO 32000-2: 20xx, Document management — Portable Document Format — Part 2: PDF 2.0

NOTE 2 This document uses the forthcoming dated revision of ISO 32000-2. This document remains under development and is only available to accredited members of ISO TC 171 SC 2 WG 8 or to members of the PDF Association. A Draft International Standard (DIS) of this document should be available for purchase from ISO in the early summer of 2019.

HTML 5, http://www.w3.org/TR/html5/

"http://www.w3.org/1999/xhtml", in "HTML5 – A vocabulary and associated APIs for HTML and XHTML; W3C Recommendation; 28 October 2014; W3C"

Cascading Style Sheets https://www.w3.org/Style/CSS/

1 Scope

This document describes an algorithm that produces conforming HTML5 from a well-tagged PDF.

It is important to see "well-tagged" in the context of known best practices for tagging that require semantic appropriateness, recommend the best use of PDF structure elements in diverse situations, and other practices.

This document identifies "well-tagged PDF" as those PDF files that conform to ISO 32000-2, 14.8 "Tagged PDF", or ISO 14289-1 (PDF/UA-1).

The best results are achieved when tagged pdf files are both authored (by users) and created (by software) with reuse in mind. In particular, the semantic structures defined in Tagged PDF are fundamental to realizing the author’s intent in the derivation context. Their presence as an accurate reflection of the author’s intent is the guarantor of an expected user experience.

This document is intended for the developer of software that:

- creates PDF files suitable for reuse
- interprets PDF contents for alternative display on mobile devices and/or HTML environments
- embeds PDF viewing into HTML pages
- derives PDF content into HTML for automated processing

This document does not:

- Provide adaptations for deriving PDF into HTML sub-structures (e.g., within a <div>)
- Provide guidance for editing or modifying PDF files or HTML derived from PDF files
- Provide guidance for addressing the security implementations of derivation
- Substitute for best-practice documents focusing on accessibility

2 Terms and definitions

derivation

deterministic process of conversion of Tagged PDF files into a syntactically valid HTML file

derived HTML

HTML produced by processors operating in conformity with this document

derived CSS

default CSS produced by processors operating in conformity with this document

media type

a two-part identifier for file formats and format contents, also known as MIME type or content type

processor

any software, hardware or other active agent that derives HTML from a tagged PDF file

tagged PDF

PDF files that conform to ISO 32000-2, 14.8 "Tagged PDF"

3 Notation

Key names are given in **boldface**, while values are given in *italics*.

In example pseudo-code, standard PDF structure element entries are given with angled-brackets (e.g., <Div>). The elements are not closed; instead, items contained within PDF structure elements are enclosed by "{ }". Attributes are indicated using HTML conventions, e.g. ‘<P lang="en-us">’, remarks or special characters are shown by [].

EXAMPLE

```
<Figure alt="PDF icon">
<Caption> {
    <P> [remark or notice]
    <P> {relevant content}
}
```

4 Algorithm for deriving HTML from Tagged PDF

This algorithm establishes requirements for processors desiring consistent results from the derivation of tagged PDF to HTML.

4.1 Technical context

Use of this algorithm assumes substantial knowledge of ISO 32000-2 in general, and subclauses 14.6 - 14.9 in particular, as well as HTML5. Format requirements in those specifications are not re-iterated here; knowledge of them is assumed.

4.2 Document handling

The processor shall initialize two output streams - one for derived HTML and one for the derived CSS. The HTML stream shall reference the CSS using conventional techniques.

> NOTE 1 The processor may decide to store derived CSS in a separate file and use a **link** element to define the reference to it in the derived HTML.

The first line of the HTML document shall be "<!DOCTYPE html>".

> NOTE 2 While not required, a DOM-like approach for both HTML and CSS document processing is recommended to allow for inline-modifications. However, the use of a stream-based approach is also acceptable.

4.2.1 Head

The first element created in the HTML output shall be a head element with four child elements, **title**, **meta**, **viewport** and **link**.

The value of the **title** element shall be derived from the value of the **dc:title** metadata value (if present) in the PDF's document-level XMP. If the PDF does not have a **dc:title** specified, the value of the **title** element in the HTML shall be derived from the PDF's filename.

All text shall be encoded using UTF-8. A **meta** element shall be added with attributes of:

- http-equiv, whose value shall be *Content-Type*
- content, whose value shall be *text/html; charset=utf-8*

A second **meta** element shall be added with attributes of:

- name, whose value shall be *viewport*
- content, whose value shall be *width=device-width, initial-scale=1*

> NOTE Using **meta** facilitates more responsive behaviour on diverse devices.

EXAMPLE

```
<!DOCTYPE html>

<html>

<head>
```

```
<title>A Document’s Title</title>

<meta http-equiv="Content-Type" content="text/html; charset=utf-
8"/>

<meta name="viewport" content="width=device-width, initial-
scale=1"/>

<link rel="stylesheet" type="text/css" href="pdf-derivation-
style.css"/>

</head>

...

</html>
```

4.2.2 The structure tree root

The structure tree root element may have one or more associated files specified via an **AF** entry. These **AF** entries shall be processed to build the **head** element of the HTML output (see 4.6, "Associated file processing").

NOTE This mechanism allows direct injection into the **head** element of an associated file of type html with a value of **Supplement** in its **AFRelationship** entry. In such a use case, it is therefore expected that the associated file is not a complete html file, but a fragment (without **head** and **body** elements) that follows HTML syntax.

4.2.3 The ClassMap

If there exists a class map dictionary (as defined by the **ClassMap** key in the structure tree root dictionary), then the processor shall iterate over all entries in that dictionary. For each entry, the processor shall add a new entry in the derived CSS file using the key name (prepended by a '.' after any escaping is expanded) as the CSS selector.

The value of each entry in the class map dictionary is an attribute object dictionary or an array of attribute object dictionaries. The processor shall identify attributes that map to CSS properties as described in 4.3.7, "Attributes", and for each, create a CSS declaration in the derived CSS using the dictionary key as the property and using the value of this key (converted into a string using common methods) as the declaration value.

If, after iterating over all attribute object dictionaries for a given key in the class map dictionary, no appropriate attributes are located, the processor may either remove the selector or provide an empty property list.

Handling the **ClassMap** in derivation is a two-step process. Attributes that represent styling are derived into a CSS style sheet and later used as a **class** attribute of the derived HTML element. Attributes that derive to HTML properties are output when processing PDF structure elements as described in 4.3.6, "Structure element properties". When an array of attribute object dictionaries is present, the processor shall respect order and process only selected attributes as described in 4.3.7, "Attributes".

EXAMPLE

PDF specifying class map

```
1  0 obj

<<

/Type /StructTreeRoot

/K [ ... ]        % PDF structure element Kids

/IDTree ...       % ID tree mapping element IDs to PDF structure
elements

/RoleMap ...      % RoleMap for the default namespace

/ParentTree ...   % Mapping for page content to parent PDF
structure elements

/ClassMap 2 0 R   % ClassMap for all elements

>>


2  0 obj                     % ClassMap dictionary

<<

/HeadingStyle

<<

/O /CSS-2.00

/text-align /center

/color /red

/font-family (Arial, Helvetica, sans-serif)

/font-size (40px)

>>


/ParaStyle

[

<<

/O /Layout

/Color [0 0 1] %blue

/BorderColor [0 1 0] %green

/TextAlign /Justify

>>


<<
```

```
        /O /CSS-2.00
        /color /red
        /font-family ("Times New Roman", Times, serif)
        /font-size (12px)
        >>
        ]
        >>
```

CSS output

```
        .HeadingStyle {
        text-align: center; color: red;
        font-family: Arial, Helvetica, sans-serif;
        font-size: 40px;
        }

        .ParaStyle {
        font-family: "Times New Roman", Times, serif;
        font-size: 12px;
        color: red; /*coming from the CSS-2.00 attribute object
        dictionary and overrides the Color attribute defined in the
        Layout attribute object dictionary*/
        border-color: green; /*coming from the Layout attribute object
        dictionary*/
        }
```

4.2.4 Body

A **body** element shall be created immediately after the **head** element. If the **Lang** key is present in the PDF's document catalog dictionary, the **lang** attribute shall be added to the **body** element with the value of the PDF document's **Lang** entry.

EXAMPLE

```
        <body lang="EN-US">
```

The children of the **body** element are created as described in 4.3, "PDF structure elements".

If the PDF contains one or more elements in the **Fields** array of the document's interactive form dictionary, then a **form** element shall be created as a child of the **body** element with an attribute, **name**, whose value shall be *acroform*.

EXAMPLE

```
<form name="acroform" id="acroform"></form>
```

All other interactive **form** elements in the document are derived to corresponding HTML form fields. They shall refer to the *acroform* using a "form" attribute of such HTML element in the derived HTML.

EXAMPLE

```
<input name="FirstName" form="acroform"/>
```

4.3 PDF structure elements

This subclause discusses processing of PDF's logical structure.

4.3.1 General

As described in ISO 32000-2, 14.7.2, PDF structure elements are constructed in a hierarchical fashion, referred to as the structure tree. Processing of the structure tree shall begin with the root element and proceed in a depth-first, pre-order traversal of each element and its children. The root element is handled according to 4.2.2, "The structure tree root".

NOTE The processing order for nodes specifically indicates pre-order for the depth-first traversal which is more explicit than logical content order.

4.3.2 Common processing

Any of the nodes in the structure tree may have one or more associated files specified via the **AF** key in the PDF structure element's dictionary. Conforming processors may use such associated files to add information to the PDF structure element's HTML output, or to replace the PDF structure element's HTML output (see 4.6, "Associated file processing").

4.3.2.1 Processing PDF structure elements

This sub-clause defines how a processor shall process PDF structure elements. Situations that require special treatment are defined in 4.3.4, "Ensuring valid HTML".

4.3.2.2 When the PDF structure element does not use an explicit namespace

If the **RoleMap** entry is present in the structure tree root, and if it contains an entry matching the structure type of the PDF structure element, the processor shall apply role mapping – possibly transitively – until no further role mapping can be applied, as described in ISO 32000-2, 14.8.6.2 "Role maps and namespaces". Based on the resulting structure type – which by definition has to be a PDF 1.7 standard structure type for any tagged PDF – the processor shall select corresponding HTML output (see 4.3.3, "Mapping PDF structure element types to HTML elements").

The processor shall add a **data-pdf-se-type-original** attribute with a value representing the original PDF structure element type before role mapping to the HTML element. If more than one role mapping is applied, the processor shall concatenate all PDF structure element types in the **data-pdf-se-type-original** attribute separated by space characters.

NOTE Extra data attributes with PDF structure types are a unified way to preserve information from PDF, and might help HTML developers to understand and rely on the original structure that would otherwise be lost during derivation.

A **data-pdf-se-type** attribute with value of the PDF standard structure type's key name shall be added to the HTML element.

EXAMPLE

PDF RoleMap definition and fragment of tagged pdf

```
1  0 obj
<<
/Type /StructTreeRoot
/RoleMap 2 0 R          % RoleMap for the default namespace
. . .
>>

2  0 obj                          % RoleMap dictionary
<<
/InlineShape /Shape
/Shape /Figure
>>
. . .
<InlineShape> {CONTENT}
```

HTML output

```
<img data-pdf-se-type="Figure" data-pdf-se-type-
original="InlineShape Shape" href="image.jpg"/>
```

4.3.2.3 When the PDF structure element uses an explicit namespace

If the PDF structure element uses either of the standard structure namespaces for PDF 1.7 or PDF 2.0 – as defined in ISO 32000-2, 14.8.6.1 "Namespaces for standard structure types and attributes" – then based on its structure type, choose an output HTML element according to "Table 1: Mapping the PDF standard structure element namespace structure types to HTML".

A **data-pdf-se-type** attribute with value of the PDF standard structure type's key name shall be added to the HTML element.

If the PDF structure element uses the MathML namespace – as defined in ISO 32000-2, 14.8.6.3 "Other namespaces" – then the processor shall use its structure type directly as a MathML element.

If the PDF structure element uses the HTML namespace the processor may use its structure type directly as the HTML element.

> NOTE 1 Direct usage of the HTML namespace raises the same security concerns that apply to HTML in general. See Annex A for additional guidance.

If the PDF structure element uses any other namespace – transitively, if applicable – the processor shall apply role mapping until encountering a structure type that belongs to one of the sets of structure types described above – PDF 1.7, PDF 2.0, MathML or optionally HTML – and then determine the HTML element to use accordingly.

> NOTE 2 This implies that not all role mappings on a given element are processed if one of the defined sets is encountered first.

4.3.3 Mapping PDF structure element types to HTML elements

Processors shall use the mappings given in "Table 1: Mapping the PDF standard structure element namespace structure types to HTML" when determining which HTML element to use when processing PDF structure element types within the PDF 1.7 and PDF 2.0 standard structure namespaces (see ISO 32000-2, 14.8.6.1, "Namespaces for standard structure types and attributes"). In many cases a straightforward mapping from PDF to HTML structure is inadequate for full conveyance of semantics; clause 4.3.5, "Special cases" provides processing requirements accommodating each of these cases.

Table 1: Mapping the PDF standard structure element namespace structure types to HTML

PDF 1.7 SSTs	**PDF 2.0 SSTs**	**HTML5 element**
Annot	Annot	- See 4.4.8.2, "Annotations (other than of type Link and Widget)". NOTE 1 This version of this document does not address the **Annot** structure element type.
Art	–	article
–	Artifact	- NOTE 2 The structure element is not output, nor is any of its content or descendent elements (see 4.3.5.7, "NonStruct, Private and Artifact").
–	Aside	aside
BibEntry	–	p

PDF 1.7 SSTs	**PDF 2.0 SSTs**	**HTML5 element**
BlockQuote	–	blockquote
Caption	Caption	caption / figcaption / div See 4.3.5.2, "Caption".
Code	–	code
Document	Document	div
–	DocumentFragment	div
Div	Div	div
–	Em	em
–	FENote	div
Figure	Figure	figure See 4.3.5.4, "Figure, Formula".
Form	Form	See 4.4.8.3, "Widget annotations".
Formula	Formula	figure See 4.3.5.4, "Figure, Formula".
H	H	h1..h6 / p Based on nesting level; see 4.3.5.1, "H, H1..Hn".
H1..H6	H1.. H6	h1..h6 / p See 4.3.5.1, "H, H1..Hn".
–	H7..Hn	p
Index	–	section
L	L	ul / ol / dl

PDF 1.7 SSTs	**PDF 2.0 SSTs**	**HTML5 element**
		See 4.3.7.4, "List standard structure attribute owner" and 4.3.5.5, "L".
Lbl	Lbl	label / span / div / dt See 4.3.5.3, "Lbl" and 4.3.7.4, "List standard structure attribute owner".
LBody	LBody	div / dd See 4.3.7.4, "List standard structure attribute owner"; see 4.3.5.5.2, "L as description list" for a description list.
LI	LI	li / div See 4.3.7.4, "List standard structure attribute owner"; see 4.3.5.5.2, "L as description list" for a description list.
Link	Link	a
NonStruct	–	- NOTE 3 The structure element is not processed, though content it contains is processed normally. See 4.3.5.7, "NonStruct, Private and Artifact".
Note	–	p
P	P	p
Part	Part	div
Private	-	- NOTE 4 The PDF structure element is not output, nor is any of its content or descendent elements. See 4.3.5.7, "NonStruct, Private and Artifact".
Quote	–	q

PDF 1.7 SSTs	**PDF 2.0 SSTs**	**HTML5 element**
Reference	-	a
RB	RB	rb
RP	RP	rp
RT	RT	rt
Ruby	Ruby	ruby
Sect	Sect	section
Span	Span	span
–	Strong	strong
–	Sub	span
Table	Table	table
TBody	TBody	tbody
TD	TD	td
TFoot	TFoot	tfoot
TH	TH	th
THead	THead	thead
–	Title	div
TOC	–	ol
TOCI	–	li
TR	TR	tr
Warichu	Warichu	span
WT	WT	span

PDF 1.7 SSTs	**PDF 2.0 SSTs**	**HTML5 element**
WP	WP	span

4.3.4 Ensuring valid HTML

PDF and HTML use different methods of expressing certain structures and restrict these structures in different ways.

To achieve interoperable reuse of PDF content in syntactically valid HTML, the derivation process has to account for these differences.

EXAMPLE

PDF allows the following as a valid nesting of standard structure elements:

```
        <Table>{
                <TR>{
                        <TH> {
                                <H1> { Heading inside TH}
                        }
                }
        }
```

As shown below, direct derivation of the above example would not produce valid HTML because the **h1** element is not allowed as a descendant of the **th** element.

HTML output

```
<table>
        <tr>
                <th>
                        <h1>Heading inside TH</h1>
                </th>
        </tr>
</table>
```

PDF allows even more complex structures that don't have a semantically equivalent expression in HTML.

EXAMPLE

PDF allows tables to include captions which may themselves include tables:

```
        <Table>{
            <TR> {..}
            <Caption> {
                  <Table> {..}
            }
      }
```

Whereas in HTML, even though the **caption** element is allowed as a descendant of a **table** element, the caption is required to be the first **table** element cannot include another **table** as its descendent.

HTML output

```
<table>
      <tr>..</tr>
      <caption>
            <table>..</table>
      </caption>
</table>
```

ISO 32000-2, 14.8.4.2 "Nesting of standard structure elements" defines rules that apply to standard PDF structure elements and the context in which they can be used.

Additionally PDF structure elements with a type of **Link** or **Form** are special cases according to 4.3.5.8, "Links and references" and 4.3.5.9, "Forms".

4.3.5 Special cases

4.3.5.1 H, H1..Hn

HTML does not directly include support for heading levels above h6, which means that **H7** and beyond PDF structure element types should typically map to **p**. To correctly convey the intended semantics, the document creator may use WAI-ARIA attributes. Processors may output such attributes automatically (even if not present in the document).

EXAMPLE

PDF

```
<H7 "O=ARIA-1.1" "role=heading" "aria-level=7" > { Heading 7 }
```

HTML output

```
<p role="heading" aria-level="7">Heading 7</p>
```

4.3.5.2 Caption

4.3.5.2.1 Captions of Figures and Formulas

If a **Caption** structure element is a direct child or an immediate sibling of a **Figure** or **Formula** structure element, then it shall be mapped to the HTML element **figcaption** and shall become the direct and first child of the corresponding HTML **figure** element.

4.3.5.2.2 Captions of Tables

If a **Caption** structure element is a direct child or an immediate sibling of a **Table** structure element, then the output HTML element shall be **caption** and it shall become the direct and first child of the corresponding HTML **table** element.

If, using this method, a **caption** element containing a **table** or **ol**/**ul** /**dl** becomes a child of another **table** element - to avoid invalid HTML, a processor may decide to:

- Move the **table** or **ol**/**ul**/**dl** sub-structure from within the **Caption** to immediately follow the parent **table**. If not allowed to be nested there continue to move up in the tree, or
- derive all PDF structure elements to span if visual representation is more critical.

EXAMPLE

Valid PDF structure without a semantic equivalent in HTML

```
<Part> {
      <Table> {
            <Caption> {
                  Some text
                  <Table> { [table inserted into the caption] }
            }
            <TR>
      }
}
```

HTML output

```
<div>
<table>
      <caption>
            Some Text
      </caption>
      <tr> </tr>
      </table>
      <table> </table>
      </div>
```

4.3.5.3 Lbl

4.3.5.3.1 Lbl within a LI (list item)

If deriving **L** to **ol** or **ul**, and if a child **LI** structure element contains a **Lbl** structure element as its first child, then:

- the **ul** or **ol** elements derived from the parent **L**'s structure element shall have an additional style attribute with value *list-style-type:none*.
- **Lbl** is mapped to **span** if it has only textual content (no other child structure elements)
- **Lbl** is mapped to **div**, if it contains other structure elements

EXAMPLE

PDF

```
      <L> {
            <LI> {
                  <Lbl> { - }
                  <LBody> {  text 1}
            }
      }
```

HTML output

```
<ul style="list-style-type:none;">
      <li><span>-</span>text 1</li>
</ul>
```

If deriving **L** to **dl**, the **Lbl** structure element is derived to a **dt** element.

4.3.5.3.2 Lbl within a Form

If a **Lbl** structure element is contained in a **Form** structure element, then:

- **Lbl** is mapped to **div** if it contains one or more of the following structure elements: **Form**, **Figure**, **Formula** or **Caption** as a direct child
- **Lbl** is mapped to **label** otherwise. If the PDF 2.0 namespace is used, an additional **for** attribute shall be added to the HTML **label** element (see 4.3.5.9.2, "Form field processing for PDF 2.0 structure elements").

4.3.5.4 Figure, Formula

If a **Figure** or **Formula** structure element is a direct child of one of **Sub, P, H, Hn, H, Em, Strong** or **Span** PDF structure element it shall not be mapped to any HTML element and the processor shall continue with its direct children, which shall themselves be mapped to **span**.

EXAMPLE

PDF

```
      <P> {
            <Figure> {
                  <Caption> {Figure Caption}
                  CONTENT [The actual image or illustration converted to
            star.jpg during derivation]
            }
      }
```

HTML output

```
<p><span>Figure Caption</span><img href="star.jpg"/></p>
```

4.3.5.5 L (list)

4.3.5.5.1 L within L

If an **L** structure element is a direct child of a **L** structure element, then the child **L** element shall be output to HTML as the direct child of a newly created **li** element.

EXAMPLE

PDF

```
      <L "ListNumbering=Ordered"> {
            <L> {
                  <LI> {Item 1.1}
            }
            <LI> {Item 2}
      }
```

HTML output

```
      <ol>
            <li>
                  <ul>
                        <li> Item 1.1</li>
                  </ul>
            </li>
            <li>Item 2</li>
      </ol>
```

4.3.5.5.2 L as description list

If an **L** structure element is derived to **dl** (see. 4.3.7.4, "List standard structure attribute owner") then its child elements shall be derived as follows:

- **LI** to **div**
- **Lbl** to **dt**
- **LBody** to **dd**

EXAMPLE

PDF

```
<L "ListNumbering=Description"> {
        <LI> {
                Lbl { First}
                LBody { the first item}
        }
        <LI> {
                Lbl {Second}
                LBody {the second item}
        }
}
```

HTML output

```
<dl>
        <div>
                <dt>First</dt>
                <dd>the first item</dd>
        </div>
        <div>
                <dt>Second</dt>
                <dd>the second item</dd>
        </div>
</dl>
```

4.3.5.5.3 L within P or Sub

If an **L** structure element is a direct child of a **P** or a **Sub** structure element the processor shall close all the HTML elements until the first parent that allows nested **ol** or **ul** or **dl** elements. The derived **ol** or **ul** or **dl** will become a child of the parent, thereafter repeating the same structure with the first sibling of the **L** element.

EXAMPLE

PDF

```
      <Part> {
            <P> {
                  <P> {
                        <P> {Actual content before the list}
                        <L "ListNumbering=Ordered">
                        <P> {Actual content after the list}
                  }
            }
      }
```

HTML output

```
      <div>
            <p><p><p>Actual content before the list</p></p></p>
            <ol>. . . </ol>
            <p><p><p>Actual content after the list</p></p></p>
      </div>
```

4.3.5.6 TH

If any heading structure element (**H, H1..Hn**) is a child of a **TH** structure element then that heading structure element shall be mapped to an HTML **p** element:

EXAMPLE

```
PDF
      <Table>{
            <TR>{
                  <TH> {
                        <H1> { Heading inside TH}
                  }
            }
      }
```

HTML output

```
<table>
      <tr>
            <th>
                  <p>Heading inside TH</p>
            </th>
      </tr>
</table>
```

If a **Sect** structure element is the child of a **TH** structure element, then all such **Sect** structure elements shall be mapped to **div** in the output HTML.

EXAMPLE

PDF

```
      <Table>{
            <TR>{
                  <TH> {
                        <Sect> {
                              <Sect> {
                                    <L> { list}
                              }
                              P {.. }
                        }
                  }
            }
      }
```

HTML output

```
<table>
      <tr>
            <th>
                  <div>
                        <div>
                              <ol> … </ol>
                        </div>
                        <p> … </p>
            </div>

        </th>

    </tr>

</table>
```

4.3.5.7 NonStruct, Private and Artifact

PDF structure elements of type **NonStruct** shall not be output to HTML, but the content they enclose (including child elements, if any) shall be processed as though it were contained in the **NonStruct** structure element's parent structure element directly.

PDF structure elements of type **Private** or of type **Artifact** shall not be output, nor shall any of their content or descendent elements.

4.3.5.8 Links and references

If the standard PDF structure element type is **Link** or **Reference**, then the HTML element shall be considered as **a**, (i.e., an HTML anchor element). The value of the **href** attribute for the HTML shall come from the annotation dictionary of the first object reference (OBJR) associated with an annotation with a **Subtype** key whose value is *Link*. If the annotation dictionary has an **A** key, and its value is an action of type *URI*, then the value of the **URI** key shall be used, or if the annotation dictionary has a **Dest** key and its value is a structured destination (ISO 32000-2, 12.3.2.3), the **id** from that destination as created according to 4.3.6, "Structure element properties" shall be used.

If a **Link** structure element is a direct child of a **Reference** structure element then the processor shall output only one HTML element with href set from the annotation dictionary represented by the **Link**.

4.3.5.9 Forms

Both the PDF 1.7 standard structure namespace and the PDF 2.0 standard structure namespace support the inclusion of form fields in the logical structure. The definition of the PDF structure element type **Form**, however, differs between the two namespaces. Accordingly, PDF structure elements of type **Form** are not derived to HTML **form** elements as such, as detailed in this subclause.

NOTE 1 HTML requires that form fields are always descendants of a form element, whereas there is no notion of an equivalent structure element in the PDF 1.7 or PDF 2.0 standard structure namespaces. Consequently, the HTML form element is inserted in a generic fashion that ensures that any PDF structure element of type **Form** will always be derived to an equivalent HTML form field that is a descendant of a **form** element.

NOTE 2 It is possible to use PDF structure elements and attributes in the HTML namespace to define forms and form fields that translate more directly into HTML elements and element structures. If form-related PDF structure elements from the PDF 2.0 standard structure namespace or the PDF 1.7 standard structure namespace on one side and from the HTML namespace on the other side were mixed inside the same document, the conversion result could be inconsistent.

4.3.5.9.1 Form field processing for PDF 1.7 structure elements

PDF structure elements of type **Form** as defined in the PDF 1.7 standard structure namespace always only contain one object reference (**OBJR**) to a widget annotation and can't contain any other content. Consequently, the derivation algorithm is based on the widget annotation.

PDF structure elements of type **Form** as defined in the PDF 1.7 standard structure namespace shall be processed as defined in 4.4.8.3, "Widget annotations".

4.3.5.9.2 Form field processing for PDF 2.0 structure elements

PDF structure elements of type **Form** as defined in the PDF 2.0 standard structure namespace always contain one object reference (**OBJR**) to a widget annotation, and can also, but are not required to, contain other content, including one or several PDF structure elements of type **Lbl**. Consequently, the derivation algorithm is based on the widget annotation, and other content inside the PDF structure element of type **Form**, with special handling of content inside PDF structure elements of type **Lbl**.

If a PDF structure element of type **Form** has descendants that are structure elements of type **Lbl**, these **Lbl** structure elements shall be created as label elements, as defined in 4.3.2.1, "Processing PDF structure elements". A **for** attribute shall be added each label element, whose value shall be the same as that of the **id** attribute of the HTML form field element created according to 4.4.8.3, "Widget annotations".

EXAMPLE

PDF

```
<Form> {

<Lbl>{Last name:}

OBJR [widget annotation of single line text field]

}
```

HTML output

```
<label for="bd43-05d-11e7">Last name:</label>

<input id="bd43-05d-11e7" type="text" name="lastname">
```

4.3.5.9.3 Form field processing for PDF structure elements from the HTML namespace

When using form field related structure elements from the HTML namespace, no processing as defined in 4.4.8.3, "Widget annotations". shall be carried out. All attributes necessary for each HTML form field must be present as structure attributes in the HTML namespace.

When using form field related structure elements from the HTML namespace, structure elements of type **form** shall be present as necessary to ensure that all form fields in the derived HTML are descendants of a **form** element as required by HTML.

4.3.6 Structure element properties

Structure element properties convey data whose processing is critical to complete and accurate conveyance of semantic meaning.

4.3.6.1 General

If the structure element dictionary contains an **ID** entry, its value shall be used as the value of the **id** attribute on the HTML element.

If a structured destination (see ISO 32000-2, 12.3.2.3) references the structure element dictionary and does not contain an **ID** entry, then a unique identifier value (generated in an implementation-dependent manner) shall be used as the value of the **id** attribute on the HTML element.

NOTE 1 This **id** is used when the **Link** annotation with the structure destination is processed.

If the PDF structure element has any classes of attributes (via the **C** key in the structure element dictionary), then those classes shall be used as the value for an attribute **class** on the HTML element. If **C** is an array, then the value of the **class** attribute shall be constructed as a concatenation of classes separated by a space character. Additionally the processor shall output attributes that map to HTML properties associated with the classes according to 4.3.7.2, "Deriving structure attributes to HTML attributes".

If the PDF structure element has an **A** key in its structure element dictionary, then its attributes shall be handled as described in 4.3.7, "Attributes", and shall be output as attributes of the HTML element or as inline styling properties.

NOTE 2 It is important to process classes of attributes before the attributes. ISO 32000-2 14.7.6.2 requires that if both the **A** and **C** entries are present and a given attribute is specified by both, the one specified by the **A** entry takes precedence.

4.3.6.2 Lang

If the structure element dictionary contains a **Lang** entry and if the entry's value is not an empty string, then its value shall be used as the value of the **lang** attribute on the HTML element.

4.3.6.3 Replacement text

If the structure element dictionary has an **ActualText** key (see ISO 32000-2, 14.9.4), contents of the key shall be used as the content of the HTML element, and the children of the PDF structure element shall be ignored.

EXAMPLE

PDF

```
<P> {
    Dru {
        <Span "ActualText=c">{k-}
        }
        ker
    }
```

HTML output

```
    <p>Dru<span>c</span>ker</p>
```

4.3.6.4 Alternate description

When processing PDF structure elements of type **Figure** or **Formula** and their structure element dictionary has an **Alt** key (see ISO 32000-2, 14.9.3), then except in those cases specified in 4.3.5.4, the contents of this key shall be used as the HTML element's **alt** attribute.

EXAMPLE

PDF

```
    <Figure "Alt=six-point star"> {
        CONTENT [The actual image or illustration converted to
    star.jpg during derivation]
    }
```

HTML output

```
    <figure><img alt="six-point star" href="star.jpg"/> </figure>
```

4.3.6.5 Expansion text

If the structure element dictionary has an **E** key that is not an empty string (see ISO 32000-2, 14.9.5), then the HTML element shall be **abbr** whose contents are the contents of the PDF structure element and a **title** attribute whose value is the UTF-8 encoded value of the expansion text.

EXAMPLE

PDF

```
    <P> {
        <Span "E=Doctor"> {Dr.}
         Jones
    }
```

HTML output

```
    <p><span><abbr title="Doctor">Dr.</abbr></span> Jones </p>
```

4.3.7 Attributes

Additional information is often associated with individual PDF structure elements through the use of structure attributes. In some cases, the presence of a specific attribute changes the selected html element, but in most cases PDF structure element attributes are mapped to HTML attributes or CSS properties.

4.3.7.1 General

Only those standard structure attributes specifically referenced in this document shall be processed. Additional format specific attributes and owners may be present, and the processor may decide to output them.

The **O** key (see ISO 32000-2, Table 376, "Standard structure attribute owners") and its value shall not be output. If the **O** key has a corresponding value of *NSO*, then the **NS** key and its value shall not be output.

Whenever an array of attributes is defined the processor shall process attributes in the following sequence:

1. List attribute owner
2. Table attribute owner
3. Layout attribute owner
4. HTML attribute owner
5. CSS attribute owner
6. ARIA attribute owner

NOTE 1 The sequence guarantees that most significant attributes are processed last.

NOTE 2 When deriving attribute values from PDF to HTML or CSS, the necessary conversion to lowercase shall be applied and only those valid in html shall be processed

4.3.7.2 Deriving structure attributes to HTML attributes

For each PDF structure element attribute mapping to an HTML attribute, the processor shall use the dictionary key as the name of an attribute on the HTML element and the value of the key (converted into a string using common methods) as the value of that attribute.

4.3.7.3 Deriving structure attributes to CSS properties

For each attribute derived to a CSS property, the processor shall create a CSS declaration using the dictionary key as the property and the value of the key (converted into a string using common methods) as the property value.

A **style** attribute for the HTML element shall be created and all CSS declarations in the current PDF structure element shall be concatenated into a string, delimited by semicolons as necessary, and the string shall be used as the value of the **style** attribute.

4.3.7.4 List standard structure attribute owner

If the list is ordered, the **L** shall be derived to **ol**. If the value of the **ListNumbering** attribute is *Description* the **L** shall be derived to **dl** (see 4.3.5.5.2, "L as description list"), otherwise it shall be derived to **ul**.

The attributes **ContinuedList** and **ContinuedFrom** shall not be processed into HTML unless an implementation is provided (e.g., equivalent CSS or JavaScript) to accommodate their semantics.

NOTE To achieve equivalent effects in an HTML, the author can provide equivalent CSS or JavaScript mechanisms.

4.3.7.5 Table standard structure attribute owner

"Table 2: Mapping Table structure type attribute owners to HTML attributes" shows the mapping from the standard table attributes to HTML attributes that shall be used by the processor when deriving **Table** structure element types to corresponding html elements.

"Table 3: Mapping standard layout attributes of Table structure elements to CSS properties" shows the mapping from the standard layout attribute belonging to **Table** structure element to CSS properties that shall be used by the processor when deriving **Table** structure element types to corresponding html elements.

Table attributes not listed in Table 2 or Table 3 shall not be processed.

Table 2: Mapping Table structure type attribute owners to HTML attributes

Standard Table attribute	**HTML attribute (output)**
ColSpan	colspan
RowSpan	rowspan
Headers	headers NOTE The mapping of the **Headers** attribute relies on the fact, that existing **ID** attributes for PDF structure elements are mapped to the **id** attribute of the **th** or **td** elements derived from **TH** or **TD** structure elements.
Scope	scope
Short	abbr

Table 3: Mapping standard layout attributes of Table structure elements to CSS properties

Standard Table attribute	**CSS property (output)**
TBorderStyle	border-style Apply any necessary conversion to lowercase

Standard Table attribute	**CSS property (output)**
TPadding	padding Apply any necessary conversion to pixels

EXAMPLE

PDF

```
<Table> {
      <TR> {
            <TH "RowSpan=2" "TBorderStyle=Dotted"> { Age }
            <TH "ColSpan=2" "TBorderStyle=Dotted"> { Names}
      }
      <TR> {
            <TH> { John }
            <TH> { Bob }
      }
      <TR> {
            <TH> { 25-30 }
            <TD> { 100 }
            <TD> { 500 }
      }
}
```

HTML output

```
<table>
<tr>
<th style="border-style:dotted;" rowspan="2">Age</th>
<th style="border-style:dotted;" colspan="2">Names</th>
</tr>
<tr><th>John</th><th>Bob</th></tr>
<tr><th>25-30</th><td>100</td><td>500</td></tr>
</table>
```

4.3.7.6 Layout standard structure attribute owner

The **TextPosition** attribute specifies whether a PDF structure element is subscript or superscript.

- If the **TextPosition** attribute is **Sup**, the PDF structure element shall map to **sup**.
- If the **TextPosition** attribute is **Sub**, the PDF structure element shall map to **sub**.

"Table 4: Mapping layout standard structure attribute owner to CSS properties" shows the mapping from the standard layout attribute to CSS properties that shall be used by the processor when deriving PDF structure element types to corresponding HTML elements.

Layout attributes not listed in Table 4 shall not be processed.

Table 4: Mapping layout standard structure attribute owner to CSS properties

Standard Layout attribute	**CSS property (output)**
Placement	If value is *Block* or *Inline*, the derived CSS property is display and values are *block* or *inline* If value is *Before*, *Start* or *End*, the derived CSS property is float with values *left* or *right*
Writing Mode	writing-mode Apply any necessary conversion to CSS property values from PDF names
BackgroundColor	background-color Apply any necessary conversion to HTMLRGB values
BorderColor	border-color Apply any necessary conversion to HTML RGB values
BorderStyle	border-style Apply any necessary conversion to lowercase

Standard Layout attribute	**CSS property (output)**
BorderThickness	border-width Apply any necessary conversion to pixels
Padding	padding Apply any necessary conversion to pixels
Color	color Apply any necessary conversion to HTML RGB values
SpaceBefore	(interpreted) There is no equivalent CSS property; the processor should use a combination of display and margin-top properties to simulate the expected behavior
SpaceAfter	(interpreted) There is no equivalent CSS property; the processor should use a combination of display and margin-bottom properties to simulate the expected behavior
StartIndent	(interpreted) There is no equivalent CSS property; the processor should use a combination of display and margin-left properties to simulate the expected behavior
EndIndent	(interpreted) There is no equivalent CSS property; the processor should use a combination of display and margin-right properties to simulate the expected behavior
TextIndent	text-indent Apply any necessary conversion to pixels

Standard Layout attribute	**CSS property (output)**
TextAlign	text-align Apply necessary conversion to CSS property values from PDF names
TPadding	padding Apply any necessary conversion to pixels
LineHeight	line-height Apply necessary conversion to CSS property values from PDF names
BaselineShift	baseline-shift Apply any necessary conversion to pixels
TextDecorationColor	text-decoration-color Apply necessary conversion to HTML RGB values
TextDecorationThickness	There is no equivalent CSS property, therefore the processor should use other properties (e.g., border-width) to achieve the same visual and semantic expression
TextDecorationType	text-decoration A *LineThrough* value shall be derived to line-through Apply necessary conversion to lowercase
RubyAlign	ruby-align Apply necessary conversion to CSS property values from PDF names
RubyPosition	ruby-position Apply necessary conversion to CSS property values from PDF names

4.3.7.7 HTML

If the value of the **O** key of an attribute object dictionary begins with the (case-sensitive) string "HTML-", then the dictionary shall be considered as containing HTML attributes and processed according to 4.3.7.2, "Deriving structure attributes to HTML attributes"..

4.3.7.8 CSS

If the value of the **O** key of an attribute object dictionary begins with the (case-sensitive) string "CSS-", then this dictionary shall be considered as containing CSS attributes and processed according 4.3.7.2, "Deriving structure attributes to HTML attributes"..

EXAMPLE

PDF

```
<H1 "O=CSS-3.00" "color=red" "font-size=12px" > { Heading 1 }
```

HTML output

```
<h1 style="color: red; font-size: 12px;">Heading 1</h1>
```

4.3.7.9 ARIA roles

If the value of the **O** key of an attribute object dictionary begins with the (case sensitive) string "ARIA-", then this dictionary shall be considered as containing ARIA attributes and processed according to 4.3.7.2, "Deriving structure attributes to HTML attributes"..

4.3.7.10 Others

Processing of attributes with any other value of the **O** key is implementation dependent and therefore beyond the scope of this document. To achieve consistent output, implementations should not override attributes defined in ISO 32000-2.

4.4 Processing of a content element

The child elements of structure elements that reference content items consist of the various types of PDF graphic objects (ISO 32000-2, 8.2): path, text, XObject, inline image and shading. Processors shall handle content items based on the use case:

- Where visual fidelity is important (infographics, charts etc.) a processor shall process content items as a group by either rasterizing all items and incorporating the result as a single raster image or by converting to SVG and include the output in the HTML. Example of such usage might be content elements within **Figure** structure element.
- For general purposes each content element object type shall be processed according to the provisions of this subclause.

4.4.1 Paths

A processor should choose one of the following methods of handling a content element that represents one or more path objects:

- simply rasterize the paths and then incorporate it into the HTML as a single raster image (see 5.2.4.4. Image XObjects and inline images), or
- convert to SVG and include it either directly in the HTML or via an **img** element, or
- represent it as a canvas object.

If the paths are irrelevant to the reuse application the processor may decide not to output path objects.

4.4.2 Text

The text of the structure content element shall be converted to UTF-8 (see 4.2.1, "Head"), and derived as the content of the HTML element.

4.4.3 Image XObjects and inline images

The image content shall be derived into an **img** HTML element. The **width** and **height** attributes on the **img** element shall be present and shall represent the logical size of the image as it would be displayed when rendering the PDF page at 100%, assuming a default viewing distance of an arm's length and page sizes typically used for reading at arm's length.

> NOTE 1 According to HTML5, width and height are specified without units and imply pixels (px). Pixels are defined in "CSS Values and Units Module Level 3" as 1/96 inch at a viewing distance of an arm's length (28 inch or 0.712 m). The values for the **width** and **height** attributes do not have to match the actual number of pixels in the horizontal and the vertical direction in the image file. If the ratio between the **width** and **height** attributes differs from the actual number of pixels in the horizontal and the vertical direction in the image file, the image will be distorted accordingly when rendered.

The manner in which image data is encoded in PDF in many regards differs from how image data is encoded in file formats such as GIF, PNG or JPEG, or in SVG. When converting from PDF image data to an OWP-supported file format, a processor should choose the most suitable file format, and should take into account the following aspects:

- the bit depth, whether by not using GIF or using dithering or other mechanisms
- the colour appearance, whether by converting to a device colour space that matches the rendering system's or device's characteristics or by embedding a suitable ICC profile
- the compression; using lossy compression only if no additional loss of information is incurred
- the effect of any **Mask** or **SMask** entries applicable to the image data in the PDF

Image XObjects that contain an **ImageMask** entry with a value of *true* shall be encoded such that the current colour in the current graphic state is taken into account, and the masking effect shall be represented appropriately in the file format to which the image is converted.

If the processor is unable to convert the data, it shall place some form of placeholder image, of the same logical (display) size, in the output HTML.

NOTE 2 This ensures that the HTML will at least layout the same way as it would if the image were present.

The value of the **src** attribute on the output **img** element shall be the URL to the image data that the processor has prepared.

NOTE 3 Since the handling of the image data is implementation-dependent, the URL can be any valid URL including absolute (with or without prefix) or data URLs (RFC 2397).

4.4.4 Form XObjects

A processor shall process a content element that represents a Form XObject as a grouping of other elements. Each of those elements shall be processed as per 4.4, "Processing of a content element".

4.4.5 Shadings

A processor should choose one of two methods of handling a content element that represents a shading:

- simply rasterize the shading and then incorporate it into the HTML as a single raster image as per 4.4.3, "Image XObjects and inline images", or
- process the shading as a vector element (path) and then address as per 4.4.1, " Paths".

If the shadings are irrelevant to the reuse application the processor may decide not to output shadings.

4.4.6 Artifacts

The derivation algorithm intentionally ignores artifacts not contained in the structure tree (see 4.3.5.7, "NonStruct, Private and Artifact").

4.4.7 Handling marked content sequences

4.4.7.1 Lang attribute in a marked content sequence

When a marked content sequence contains the **Lang** attribute, the content enclosed by this marked content sequence shall be enclosed in a **span** element having a **lang** attribute whose value is the UTF-8 encoded value of the **Lang** attribute.

4.4.7.2 ActualText attribute in a marked content sequence

When a marked content sequence contains the **ActualText** attribute, the content enclosed by this marked content sequence shall be replaced by the UTF-8 encoded value of the **ActualText** attribute and shall be enclosed in a **span** element.

4.4.7.3 Alt attribute in a marked content sequence

When a marked content sequence contains the **Alt** attribute, the content enclosed by this marked content sequence shall be enclosed in a **span** element having an **alt** attribute whose value is the UTF-8 encoded value of the **Alt** attribute.

4.4.7.4 E attribute in a marked content sequence
When a marked content sequence contains the **E** attribute, the content enclosed by this marked content sequence shall be enclosed in an **abbr** element having a **title** attribute whose value is the UTF-8 encoded value of the **E** attribute.

4.4.7.5 Multiple attributes in a marked content sequence
When a marked content sequence contains more than one of the **Lang**, **ActualText**, **Alt** or **E** attributes, only one **span** element shall be created. If the **E** attribute is one of these attributes, the **abbr** element shall be created inside the span element, with the content inside the marked content sequence or, in the case where an **ActualText** attribute is present, the UTF-8 encoded value of the **ActualText** attribute as its content.

4.4.8 Processing of an object reference (OBJR)

4.4.8.1 XObjects
Object references in structure elements of type **XObject** shall be processed according to 4.4.4, "Form XObjects".

4.4.8.2 Annotations (other than of type Link and Widget)
Handling of annotations other than Links and Fields/Widgets will be addressed in a future version of this specification.

NOTE All other annotation types are out of scope for this document.

4.4.8.3 Widget annotations
Object references in structure elements of type **Form** reference widget annotations. Based on the type of the form field it belongs to, a widget annotation will be processed differently.

HTML provides different types of elements for different types of form fields, such as **button**, **input**, **select** and **textarea**, which are collectively referred to as HTML form fields.

Widget annotations that are invisible or hidden, have a width or a height of *0* (zero), or are completely outside the **CropBox** – or in the absence of the **CropBox,** completely outside of the **MediaBox** – of the page on which they are present, or are not present on any page, shall be processed with CSS property **display** set to **none**

4.4.8.3.1 Mapping widget annotations to HTML
Widget annotations shall be mapped to one of the following HTML elements. Additional HTML attributes and inner HTML shall be derived as defined in the following tables.

- Button (see "Table 5: Mapping widget annotations to button HTML elements")
- Input (see "Table 6: Mapping widget annotations to input HTML element")
- textarea (see "Table 7: Mapping widget annotations to the textarea HTML element")
- select (see "Table 8: Mapping widget annotations to select HTML element")

Table 5: Mapping widget annotations to button HTML elements

Type of field	**type attribute**	**Additional attributes**
Push button field	button	
Submit button (Push button with **A** (action) entry where the **S** (subtype) entry's value is *SubmitForm*); The **ExportFormat** flag shall be set to *HTML*	submit	Map **URL** in **F** in *SubmitForm* action to **formaction** attribute Map **GetMethod** flag to **formmethod** attribute with value *get* or *post*
Reset button (Push button with **A** (action) entry with the **S** (subtype) entry's value is *ResetForm*)	reset	
Import-data button (Push button with **A** (action) entry with the **S** (subtype) entry's value is *ImportData*)	button	button NOTE Import-data is out of scope for this document; if encountered it is processed like a regular Push button field

If the derived HTML element is **button**, then inner HTML shall be created with

- N appearance stream per 5.2.4. Processing of a content element
- CA entry from MK dictionary

Table 6: Mapping widget annotations to input HTML element

Type of field	**type attribute**	**Additional processing**
Check box button field	checkbox	If an **Opt** entry is present, map the applicable entry to the **value** attribute. If an **Opt** entry is not present, map the name in the Widget's normal

Type of field	**type attribute**	**Additional processing**
		appearance stream (as defined by a value other than *Off* in the **N** dictionary of the widget's **AP** dictionary), to the **value** attribute. If the **AS** entry's value is not *Off*, set the **checked** attribute
Radio button field NOTE: The flag **RadiosInUnison** is not supported.	radio	If an **Opt** entry is present, map the applicable entry to the **value** attribute. If an **Opt** entry is not present, map the name in the Widget's normal appearance stream (as defined by a value other than *Off* in the **N** dictionary of the widget's **AP** dictionary), to the **value** attribute. If the **AS** entry's value is not *Off*, set the **checked** attribute
Single line text field	text	If the **RichText** flag is not set and **RV** is not present, map **V** to **value** Map **MaxLen** to **maxlength** Map **DoNotSpellCheck** to **spellcheck** If the **RichText** flag is set and **RV** is present, additional inner HTML from the **RV** entry shall be created.
Password text field (i.e. Single line text field with the **Password** flag set; multiline text fields with **Password** flag set are not supported, and will be mapped as single line text fields)	password	Map **V** to **value** Map **MaxLen** to **maxlength** Map **DoNotSpellCheck** to **spellcheck**

Type of field	**type attribute**	**Additional processing**
File select text field (i.e. Single line text field with the **FileSelect** flag set; multiline text fields with **FileSelect** flag set are not supported, and will be mapped as single line text fields)	file	Map **V** to **value** Map **MaxLen** to **maxlength** Map **DoNotSpellCheck** to **spellcheck**
Choice field with Edit flag set	text	Map **V** to the **value** Add **list** attribute referring to an **id** of the associated **datalist** element (see below) Create sibling **datalist** with a unique **id** property Map **Opt** array values to inner **option** elements inside **datalist** NOTE As of today, **datalist** is not supported in IE9 or earlier or in Safari.

Table 7: Mapping widget annotations to the textarea HTML element

Type of field	**Additional processing**
Multiline text field	Map **MaxLen** to **maxlength** Map **DoNotSpellCheck** to **spellcheck** If **RichText** flag is set and **RV** is present, inner HTML from **RV** entry shall be created; otherwise create inner HTML from **V** entry

Table 8: Mapping widget annotations to select HTML elements

Type of field	**Additional processing**

ListBox	Set **size** to 3
Combo	

If the derived HTML element is **select**, then:

- If **Multiselect** field is defined, add **multiple** HTML element
- Map the entries from the **Opt** entry of the form field to **option** inner HTML
- Map **V** and **I** to the attribute(s) **selected** in the corresponding **option** element(s)

4.4.8.3.2 Widget annotation attributes

Certain widget annotation attributes (see ISO 32000-2, "12.5.6.19 Widget annotations"), if present, shall be added to the HTML form field element:

As local style attributes, using suitable CSS declarations:

- highlighting mode (**H** entry)
- border style (**BS** entry)
- border color (**BC** entry in the MK dictionary)
- background color (**BG** entry in the MK dictionary)
- text alignment as defined in the **Q** entry if applicable for the derived HTML element

As HTML attributes:

- ReadOnly (**Ff** entry) mapped to readonly
- Required (**Ff** entry) mapped to required
- The fully qualified form field name (as defined in ISO 3200-2, 12.7.4.2 "Field names") mapped to name

4.5 ECMAScript

To achieve an equivalent experience in HTML as when processing forms in the PDF context, the processor shall derive embedded ECMAscripts into HTML javascript when deriving Widget annotations into HTML form fields. ECMAScript for PDF (see ISO 21757-1) defines the set of static and dynamic objects available to PDF.

The recommended way is to develop a JavaScript library which provides implementations of the ECMAScript objects. The implementation details are not part of this specification; it's up to the developer to ensure the expected behavior. See Annex A "ECMA script derivation guidance" for examples of implementation.

4.6 Associated file processing

4.6.1 General

Each associated file's file specification dictionary may either refer to an embedded file stream or an external URL-based reference. If the file specification dictionary contains an **FS** key with a value of *URL* and does not contain an **EF** entry, then it shall be handled as "URL References" as described in all sub-clauses of 4.6, "Associated file processing". If the file specification dictionary contains an **EF** entry, then it should be processed as "Embedded Files" as described in all sub-clauses of 4.6, "Associated file processing". The processor shall ignore all other file specification dictionaries.

While it is recommended to process associated files as described in this chapter, the implementer may decide not to do so, or limit implementation only to certain media types due to security concerns. See Annex A.

4.6.2 URL References

For URL References, the value of the **F** entry in the associated file's file specification dictionary is the URL that shall be used to refer to the external services. URL References shall not target local files nor make use of the file URL scheme.

NOTE 1 File URL schemes are specified in RFC 1738, Uniform Resource Locators (URL). The prohibition of file URL schemes implies that it is not possible to reference local files.

For Embedded Files, the URL shall be the value of the **UF** entry from the associated file's file specification dictionary.

NOTE 2 This requirement ensures that resources and associated files can reliably refer to each other, for example CSS referring to an image to be used as a background.

4.6.3 Media types

The handling of an associated file, whether it is a URL Reference, or an embedded file shall be based on its media type.

For URL References, the filename extension of the URL (see 4.6.2, "URL References") shall be used in conjunction with "Table 9: Media types supported by embedded files " to determine the media type of the associated file.

For embedded files, the media type shall be determined by the value of the **Subtype** key of the embedded file stream dictionary that is the value of the **EF** key of the associated file's file specification dictionary.

"Table 9: Media types supported by embedded files" lists the known media types, their filename extensions, what each represented and which of the following sub-clauses provides more information about processing it.

If the file extension of the associated file is not one of the known extensions corresponding to the media types specified in "Table 9: Media types supported by embedded files " then the processor may process it or ignore it as it deems appropriate. A processor may support additional filename extensions and/or media types beyond those in the table.

Table 9: Media types supported by embedded files

Media types	**Filename extensions**	**Type of object**	**Sub-clause**
text/html application/xhtml+xml	.htm, .html, .xhtml	HTML or XHTML	4.6.4.2
text/css	.css	CSS	4.6.4.3
text/javascript application/javascript	.js	JavaScript	4.6.4.4
image/jpeg image/png image/gif	.jpg, .jpeg, .png, .gif	Images	4.6.4.5
image/svg+xml	.svg	SVG	4.6.4.6
application/mathml+xml	.xml, .mathml	MathML	4.6.4.7

4.6.4 Handling media types

4.6.4.1 General

When processing a structure element with an associated file, in some cases the associated file will replace the otherwise generated HTML element while in others it will be additive:

- If the value of the **AFRelationship** key in the associated file's file specification dictionary is *Alternative* then the associated file serves as a replacement and all children of the structure element shall be ignored.
- If the value of the **AFRelationship** key in the associated file's file specification dictionary is *Supplement* then the associated file serves as a supplemental and after processing the associated file the processor shall continue with processing children of the structure element.

In both cases all requirements for attribute processing (see 4.3.7, "Attributes") shall apply.

NOTE This enables an author to provide specific attributes on the output HTML elements by having them present on the PDF structure element.

Associated files with a value other than *Alternative* or *Supplement* for the **AFRelationship** key in the associated file's file specification dictionary may be ignored; the processor shall continue with children of the structure element.

Multiple associated files shall be processed in the order in which they are stored in the array of the **AF** key.

For security reasons, processors may choose to mitigate risks by ignoring categories of Associated Files.

4.6.4.2 HTML

If the associated file is an URL Reference, then the processor shall add a **link** element to the **head** element of HTML output, with attributes of **rel** (with a value of *import*) and **href** (with a value that is the URL).

If the associated file is an Embedded File then the contents of the associated file's embedded file stream shall be added directly to the output HTML stream, taking the place of the structure element that would normally have been generated.

> NOTE This mechanism allows direct injection of an associated file of type HTML with **AFRelationship** of *Supplement* into the output HTML stream. It is therefore expected that the associated file is not a complete HTML file, but a portion that follows HTML syntax.

4.6.4.3 CSS

If the associated file is either a URL Reference or an Embedded File of type CSS, then the processor shall add to the output HTML, immediately before the referencing HTML element, a **style** element, whose contents shall consist of an **@import** declaration with a value of the URL.

EXAMPLE

```
<style>@import url(specialtable.css);</style>
```

4.6.4.4 JavaScript

If the associated file is either a URL Reference or an Embedded File of type JavaScript, then the processor may add to the output HTML, immediately after the referencing HTML element's closing tag, a **script** element with an attribute of **src** whose value is the URL and no contents.

EXAMPLE

```
<script src="specialtable.js"> </script>
```

If the structure element with the associated file attached derives to **script** in the HTML namespace (http://www.w3.org/1999/xhtml), then the HTML element shall be **script**. All children of the structure element shall be ignored.

4.6.4.5 Images

To incorporate images into the HTML output, regardless of whether the associated file is a URL Reference or an Embedded File, an **img** element shall be added to the HTML output with a **src** attribute whose value is the URL.

4.6.4.6 SVG

To incorporate SVG into the HTML output, regardless of whether the associated file is a URL Reference or an Embedded File, an **img** element shall be added to the HTML with an attribute of **src** whose value is the URL. If the structure element has a BBox structure attribute (of any owner or namespace), then the height and width of that BBox shall be written out, respectively, as **height** and **width** attributes on the **img** element. These **height** and **width** attributes should be determined as described in 4.4.3, "Image XObjects and inline images".

4.6.4.7 MathML

If the associated file is an Embedded File containing MathML then the contents of its embedded file stream shall be added directly to the HTML output, taking the place of the structure element that would normally have been output.

> NOTE Since MathML is not supported by all user agents, a conforming processor may need to take additional steps to ensure that it is presented as the author expected.

Annex A: Security implications

(informative)

There are serious security concerns when it comes to derivation of PDF files to HTML. PDF structures may contain information that can take advantage of the derivation process and embed malicious code into derived HTML. One major concern is the fact that PDF files may contain such code, and the process of derivation defined in this document does not guarantee full control over output HTML. In the case of a public service that allowed users to upload PDF files in order to experience in HTML form through derivation, an attacker could leverage this case by uploading crafted PDF; derivation in itself does not prevent creation of malicious HTML.

Examples of such scenarios may include:

- Embedded JavaScript could access a whole web page if the PDF is derived into a <div>, facilitating the delivery of malicious information
- JavaScript could access cookies

It is therefore the responsibility of the developer to recognize security risks in each specific implementation. While using derivation in an enclosed environment where the developer controls the HTML viewing system, the risk might be considered as low. In cases such as, allowing users to upload random PDF files to be served as HTML to other users or systems, the developer should clearly apply stringent processing requirements.

Annex B: ECMAscript derivation guidance

(normative)

It is not in the scope of this document to define precisely how PDF ECMAscript shall be derived into JavaScript libraries for use with HTML. In this Annex we will provide guidance and examples focusing on the most common functionality.

EXAMPLE **app** object represents the application, in desktop environment the application works with several opened documents available through **activeDocs** property or require interactivity with end user through the **alert** method. Desired functionality might be different in an HTML environment and method **activeDocs** could always return 1 and **alert** method could be implemented with **window.alert()** or **with console.log()** function.

A minimal **app** implementation could look like following code:

```
var app = new Object();
//properties
app.viewerVersion = 1;
app.viewerType = "Derivation";
//methods
app.response = function () { return null; };
app.beep = function (b) { };
app.alert = function (msg) {
window.alert(msg);
};
```

Each HTML form field should have its own **Field** JavaScript object that mimics the source ECMAScript object.

It is recommended to create a **Field** object only when the HTML form field is used or required; creating and maintaining the array of all fields as appropriate. Fields are identified by name as required by ISO 32000-2, 12.7.4.2 "Field Names".

EXAMPLE The following _init function is invoked when the HTML file is loaded by calling:

```
 document.addEventListener("DOMContentLoaded", _init);
function _init() {
  var elems = document.getElementsByTagName("input");
  for (var i = 0; i < elems.length; i++) {
      e.addEventListener("focus", field_event);
      e.addEventListener("change", field_event);
      e.addEventListener("click", field_event);
      //only push when elems[i] doesn’t exist in the all_fields array
all_fields.push(elems[i]);
  }
  // the same for "select", "textarea"
  do_calculations();
}
function field_event(e) {
//checks the array of all fields if the field with the name exists.
returns existing or creates new one
  var f = init_field(i.e., target.name);
      . . .
  // keypress - focused text edit
  if (e.type == "keypress") {
      var keyCode = 0;
      if (e.keyCode != undefined && e.keyCode >= 20)
                 keyCode = e.keyCode;
else if (e.charCode != undefined && e.charCode >= 20)
                 keyCode = e.charCode;
      if (keyCode != 0)
                 event.change = String.fromCharCode(keyCode);
     event.selStart = e.target.selectionStart;
     event.selEnd = e.target.selectionEnd;
  }
. . .
// similarly, for "change" "click" etc.
 . . .
//process the event on the field, check results do calculations return
status
…
  return result;
}
// make sure the implementation is consistent and accessed fields
through ECMA Script follow the same pattern
this.getField = function (name) {
  return init_field(name);
```

```
};
```

One ECMAScript **Field** object may reference more widget annotations; the same functionality shall be preserved in derivation to HTML:

- When ECMAScript changes a value, all HTML form fields with the same name shall change their value.
- When one HTML form field is changed, the corresponding **Field** object is changed together with all related HTML form fields.

The processor shall include all document level ECMAScript methods as defined by the **JavaScript** entry in the **Names** entry in the document catalog dictionary and ECMAScript page level events defined by the **AA** entry in page dictionary.

When deriving the widget annotation, the processor shall expand the JavaScript library with methods that are defined for each form field in the form field's additional-actions dictionary. See ISO 32000-2, Table 199: Entries in a form field's additional-actions dictionary.

NOTE 1 It is best practice to generate function names for each field's method based on field identifier, which makes managing the invocation of functions as easy as possible.

Processors should keep all calculated fields in a separate array to have the do_calculation method optimized.

NOTE 2 HTML form fields always shows formatted value, while real value is preserved in the **Field** object.

Bibliography

RFC 1738, *Uniform Resource Locators (URL)* (December, 1994) Internet Engineering Task Force (IETF)

Tagged PDF Best Practice Guide 1.0, PDF Association

NOTE: Publication of this document is expected in Q2 of 2019

Matterhorn Protocol 1.02 (April, 2014), PDF Association