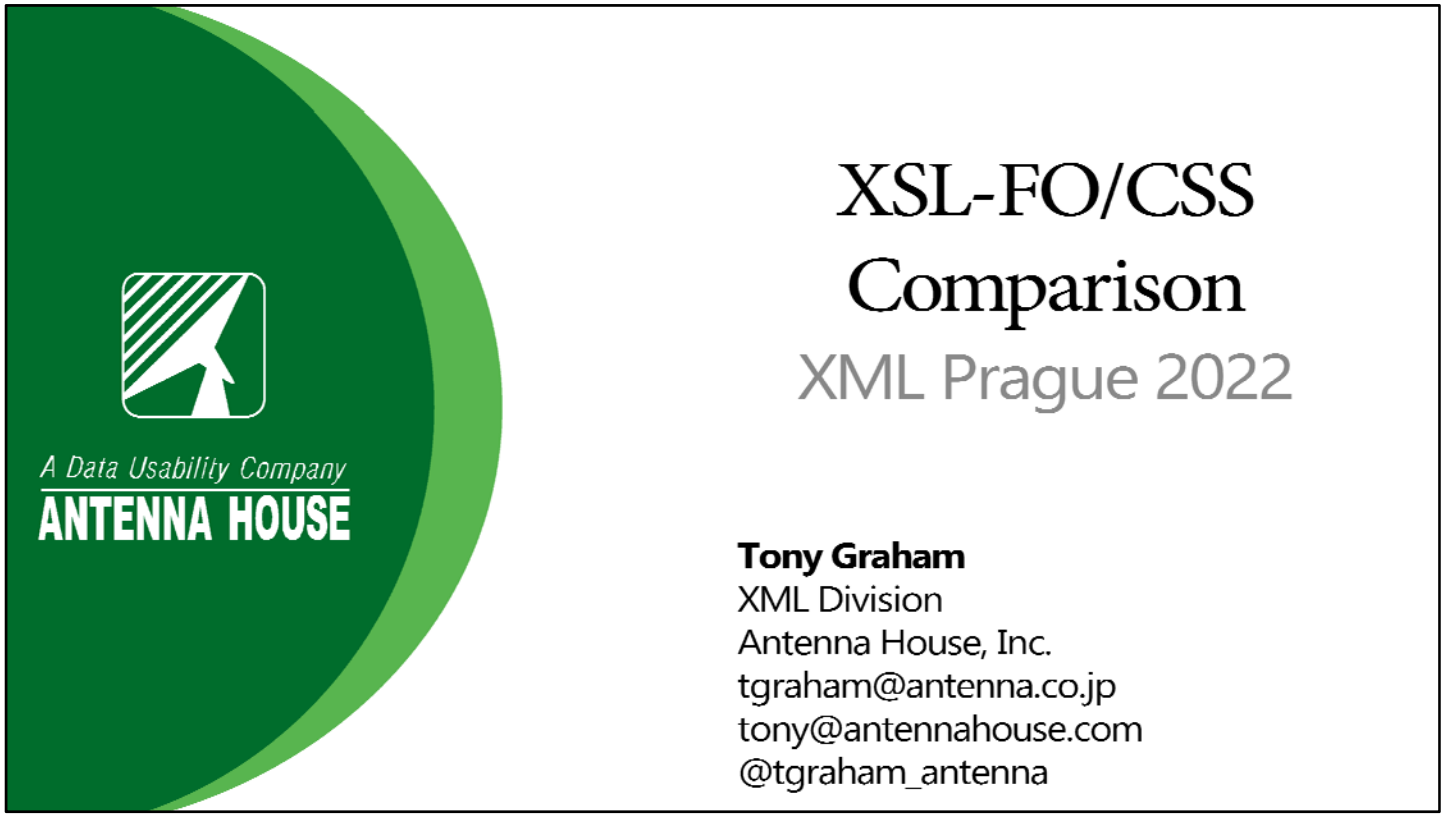
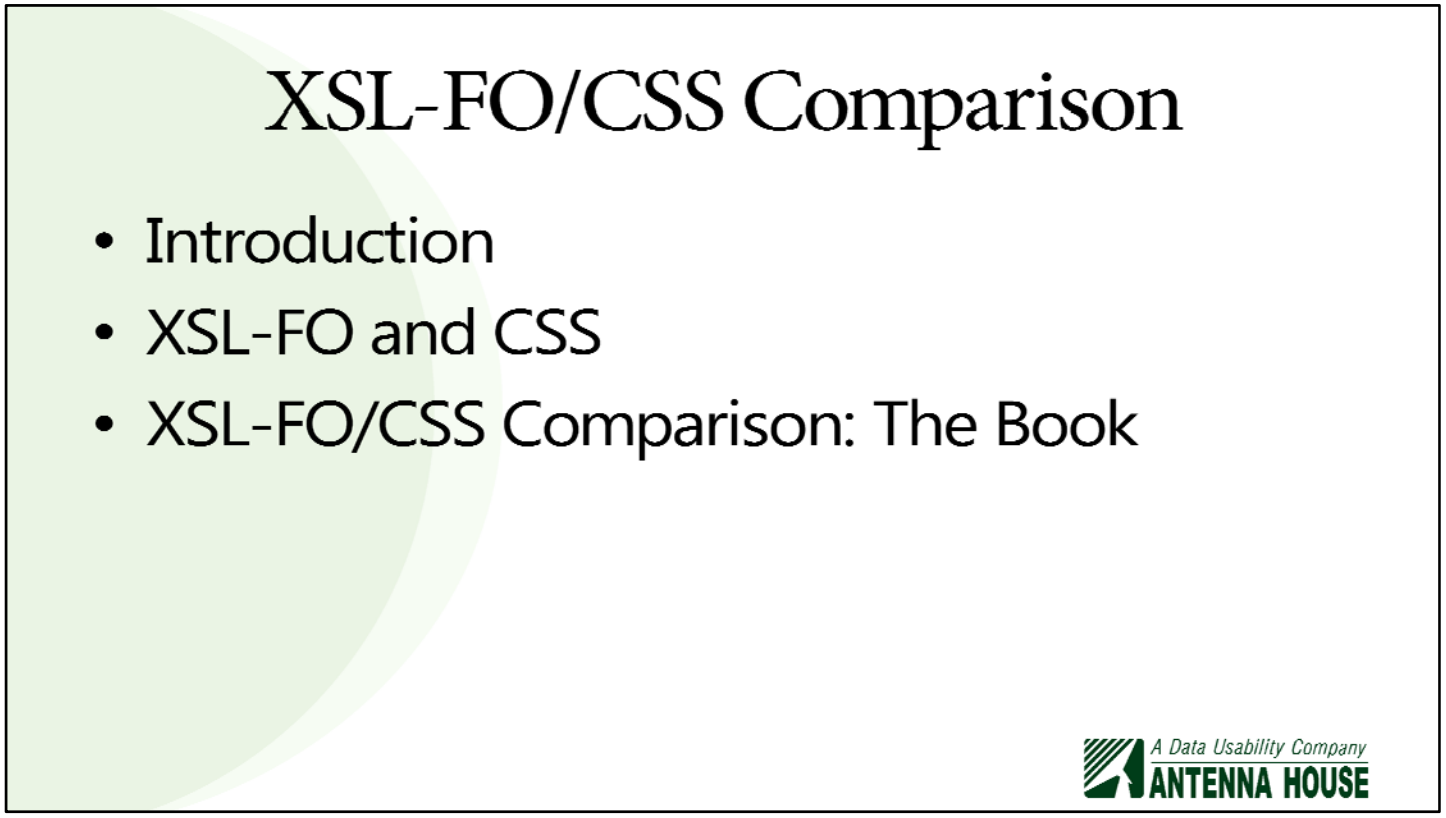
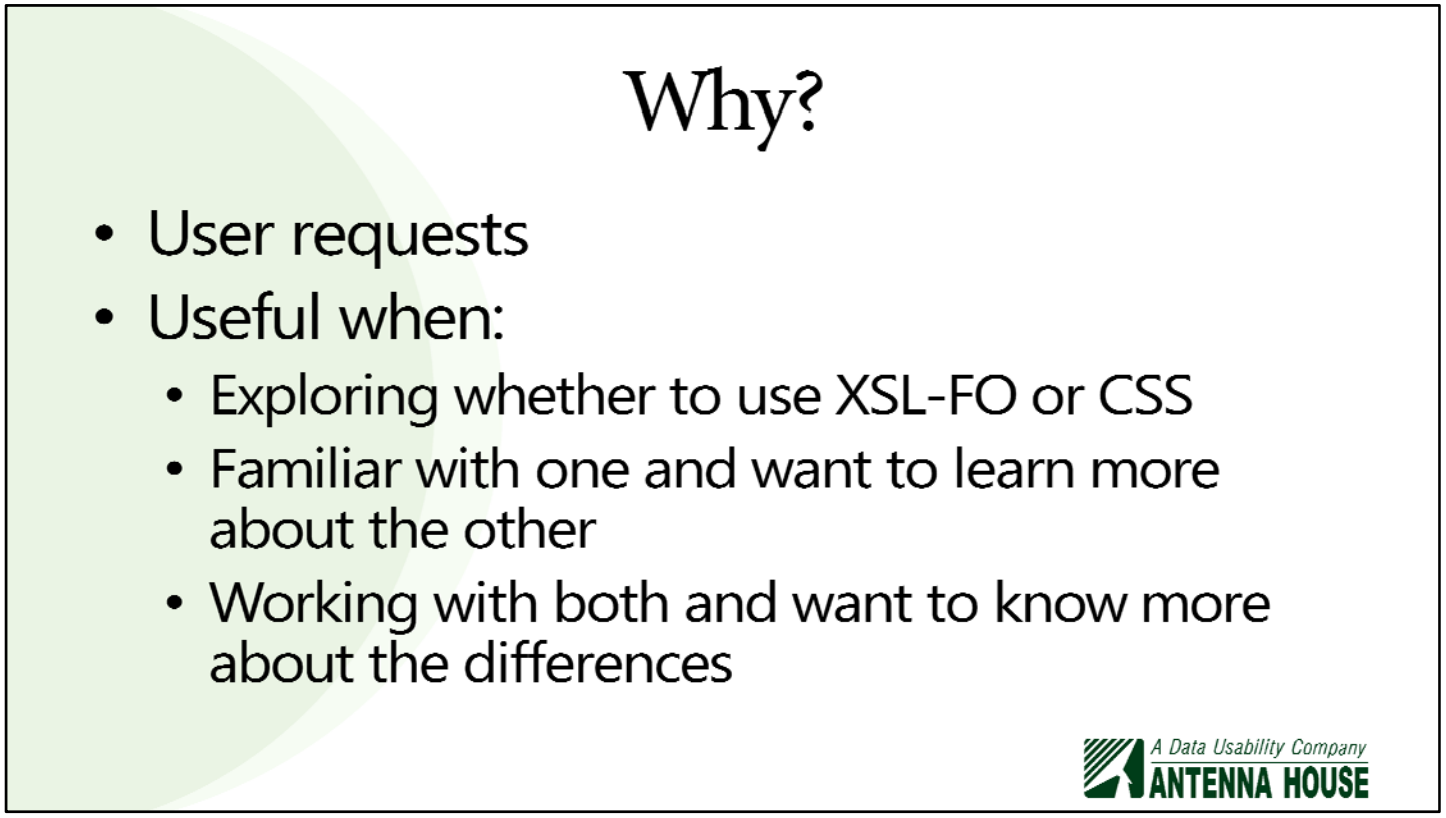


Hello, I am Tony Graham from Antenna House.

Today's talk is based on a new *XSL-FO/CSS Comparison* book produced by Antenna House. That book is available in two versions – styled using XSL-FO and styled using CSS – that are so similar that you need the Antenna House Regression Testing System (AHRTS) to see the differences.



After this introduction, this becomes a talk with two parts: a discussion of the similarities and differences between XSL-FO and CSS followed by some details about how the new 150-page *XSL-FO/CSS Comparison* book was produced.

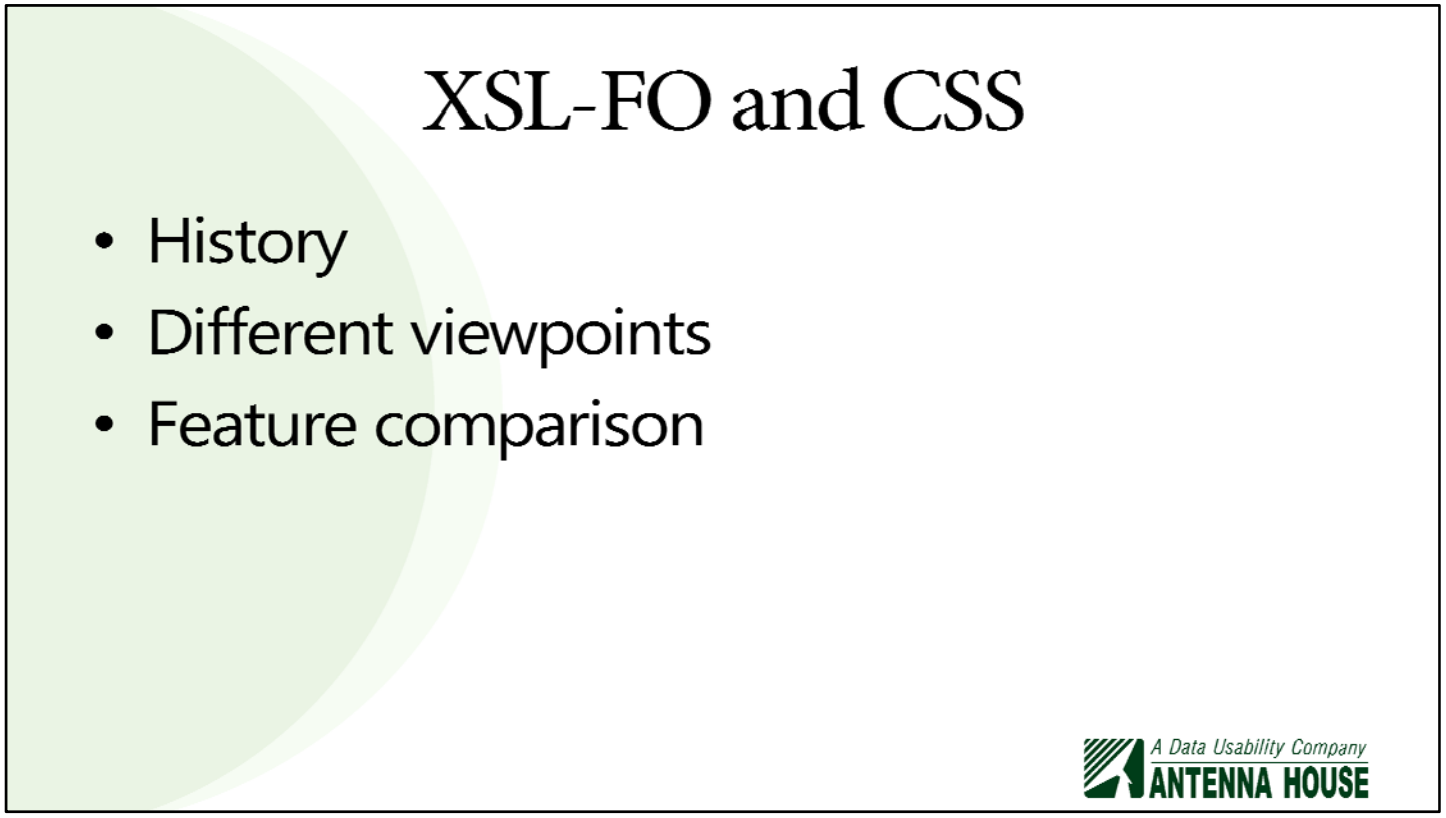


Why did we create the *XSL-FO/CSS Comparison*?

It started because of user requests to know about the similarities and differences between XSL-FO and CSS.

Initially there was a list, which became a table much like the table in the paper in the proceedings, and now it is a book.

The comparison is useful when:
- You are exploring whether to use XSL-FO or CSS
- You are already familiar with one technology and want to learn more about the other
- You are working with both and want to know more about the subtler differences between the two



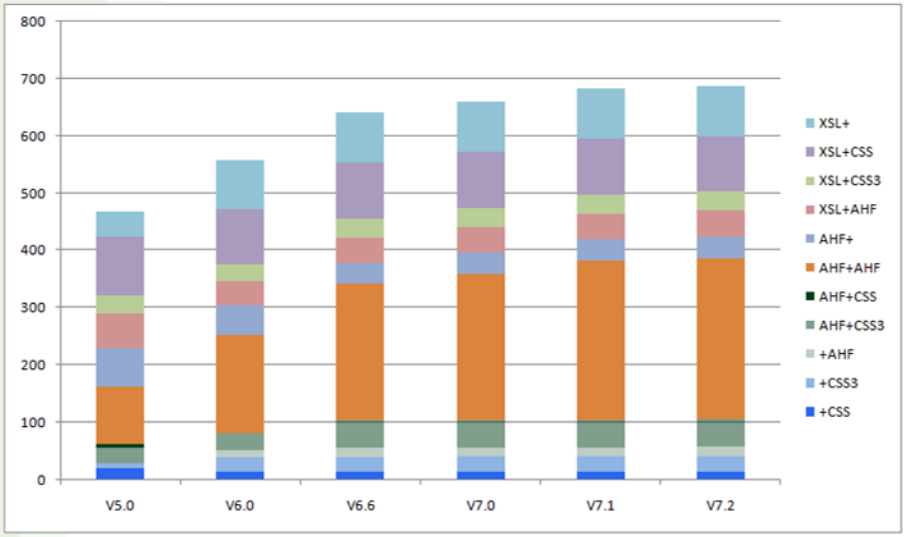
Today's comparison covers three aspects:
- The intertwined history of XSL-FO and CSS
- The differing viewpoints of a user of one technology towards the other
- A comparison of some of the differences between XSL-FO and CSS

History

Year	XSL-FO	CSS	HTML
1996	DSSSL	CSS 1	
1998		CSS 2	XHTML 1.0
1999		First CSS 3 drafts	
2001	XSL 1.0		XHTML 1.1
2004			WHAT WG formed
2006	XSL 1.1, XSL-FO 2.0 Workshop		HTML WG rechartered
2009	XSL-FO 2.0 Design Notes		
2011		CSS 2.1	
2018		CSS Snapshot 2018	
2019			W3C cedes HTML5 to WHAT WG
2020		CSS Snapshot 2020	

- CSS 1, which became a W3C Recommendation in 1996, built on previous proposals by its co-inventors, Bert Bos and Hakon Wium Lie, among others.
- The stated goals of CSS 1 include “CSS supports stream-based (or incremental) formatting where possible” and “offers both readers and authors control over style”.
- Also in 1996, DSSSL, the style component of SGML, became an ISO standard. DSSSL included both transformation and formatting. The most popular DSSSL engine just did formatting, although it later developed its own transformation method.
- XML, from the beginning, was meant to have a style component. In 1997, Jon Bosak, original XML Working Group (WG) Chair, referred to it as “xml-style (Part 3 of the XML specification suite)” and “Part 3 of the W3C XML suite of specifications for the use of SGML, HyTime, and DSSSL subsets on the World Wide Web”.
- xml-style eventually became XSL as we know it. XSL 1.0 became a Recommendation in 2001, and XSL 1.1, in 2006.
- XSL includes both transformation and formatting, just like DSSSL, but the transformation part was generally useful, so it was broken out as XSLT.
- In 2002, a TAG finding stated that consistency in properties was one of the architectural principles of the Web.
- In 2006, at the XSL-FO 2.0 Workshop in Heidelberg, Antenna House proposed compatibility between XSL-FO and CSS 3, but the proposal was voted down.

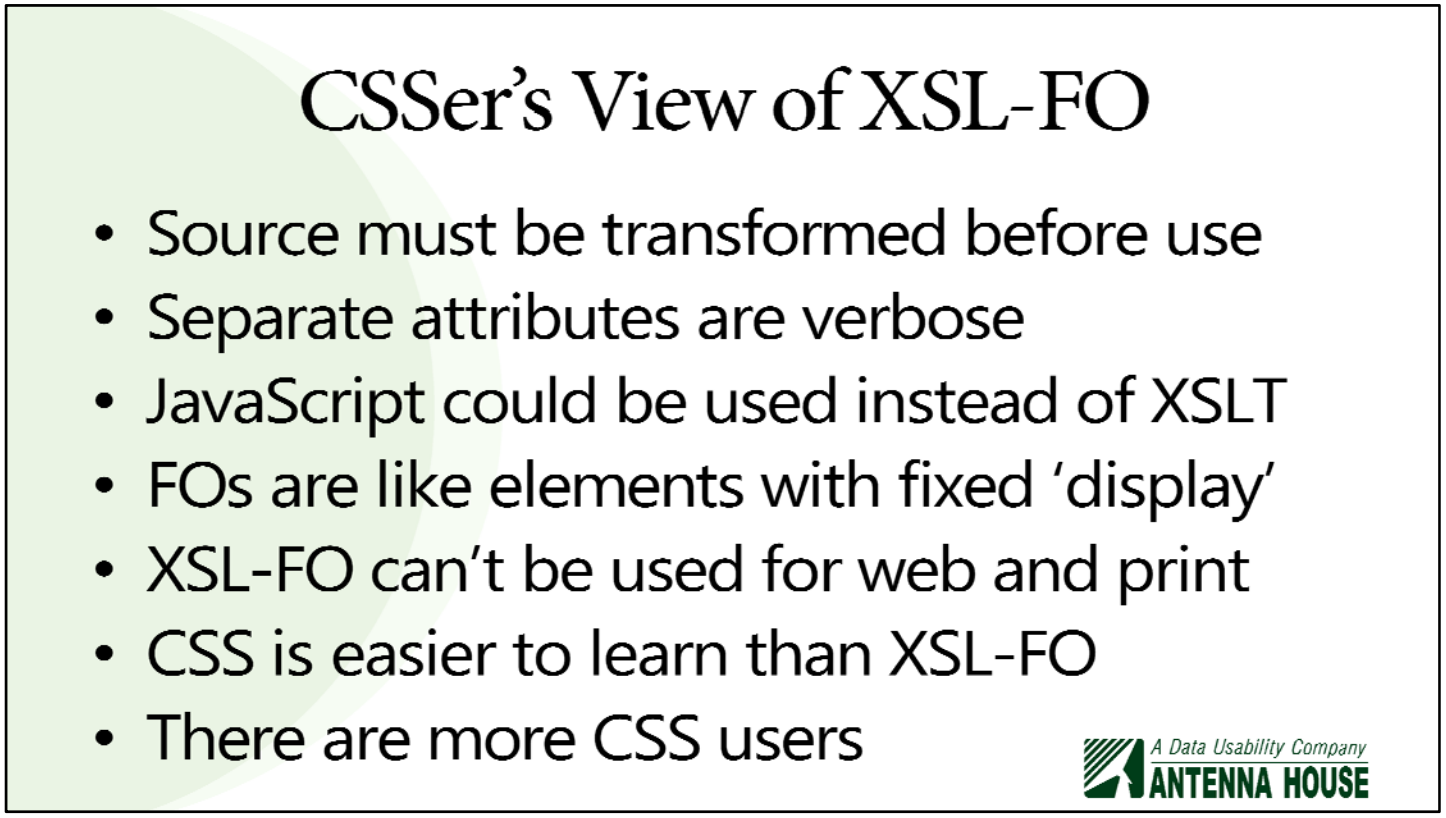
XSL-FO and CSS Properties



A Data Usability Company
ANTENNA HOUSE

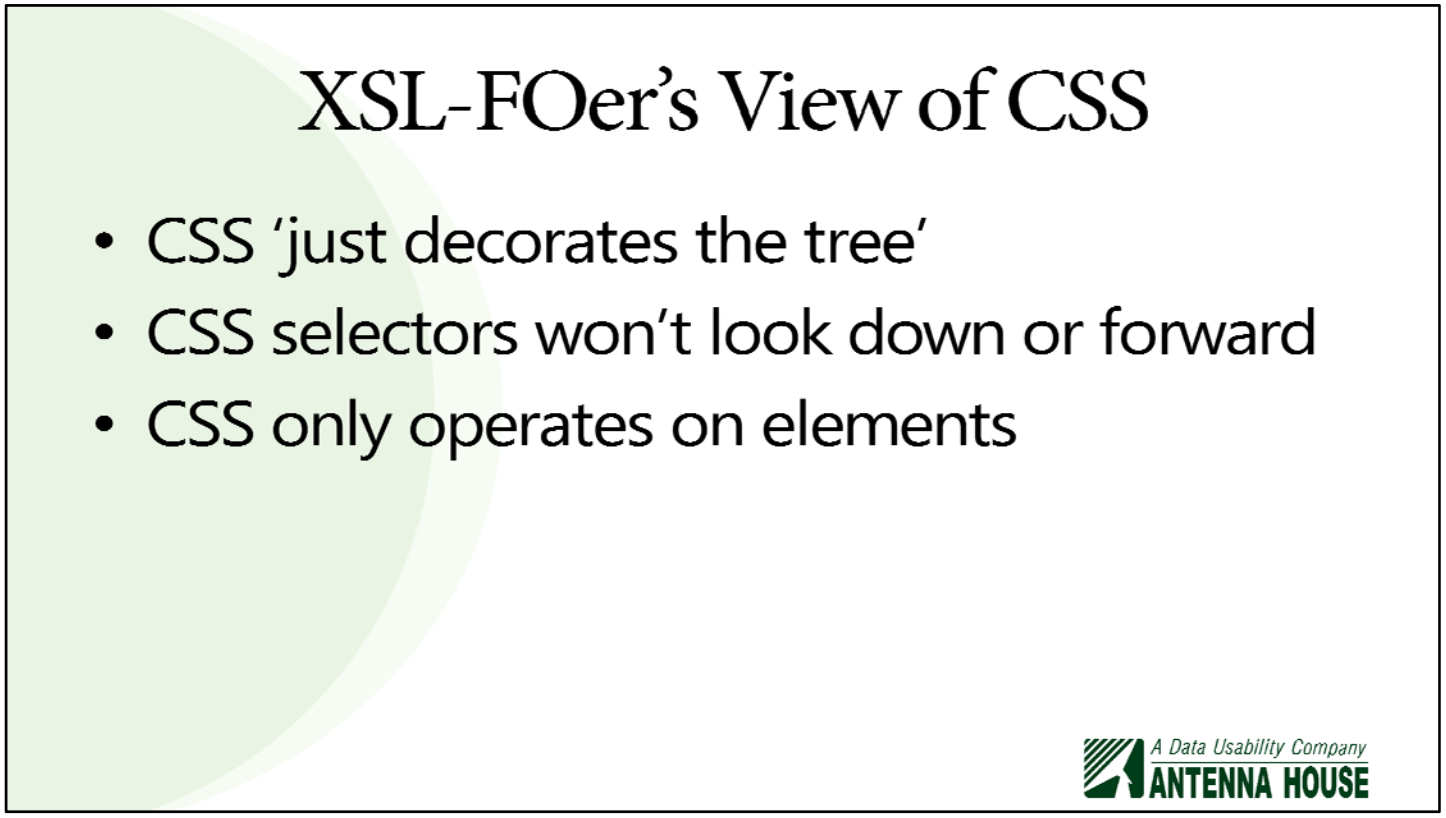
It is hard to find how XSL-FO and, particularly, CSS properties have changed over time, so I had to use the properties implemented by AH Formatter as a proxy for the number of paged media properties in XSL-FO and CSS.

If you look at the purple 'XSL+CSS' segment in each bar, you can see there are a number of properties that are common to both XSL-FO and CSS. There are more of those than there are XSL-FO-only properties or CSS-only properties.



This is an informal summary of how a CSS user might see XSL-FO.

- Yes, XSL-FO requires that the source is transformed into XSL-FO, but many CSS Paged Media documents require transformation of the source to generate tables of contents, indexes, and content that will be extracted from the flow and used in headers and footers.
- A separate attribute for each property is verbose, but it works well with XSLT.
- JavaScript could be used instead of XSLT, but AFAIK there isn't a standard JavaScript transformation language that works as well as XSLT does.
- CSS needs the 'display' property because everything is formatted on the fly. The transformation into XSL-FO makes those decisions.
- XSL-FO can express pages with unbounded dimensions, but there has been little call for it.
- The basic structure of CSS as selectors, rules, properties, and values is easy to learn, but the 2014 charter for CSS describes CSS as "a large and complex language". Later charters reframed it as a "large language" even though its complexity has increased since 2014, not reduced.
- There are more CSS users, but only a small proportion of those know much about CSS Paged Media.



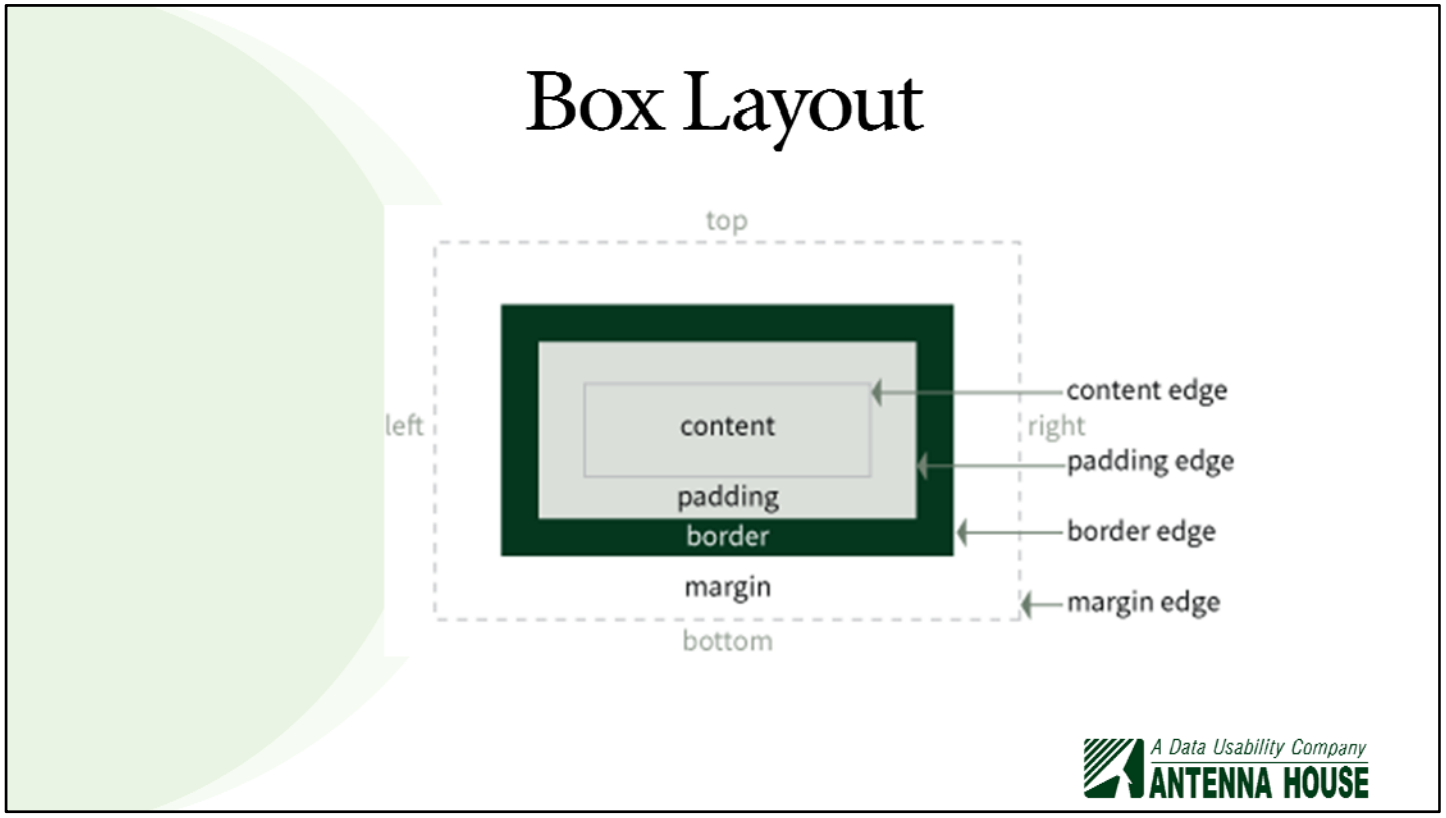
- CSS just decorates the tree because one of its goals is to be streamable and support incremental formatting. However, as stated previously, the tree of the source document often needs to be augmented to add tables of contents, indexes, and header and footer content.
- Selectors not looking down or forward also follow from CSS being designed to be streamable.
- That CSS only operates on elements can be one of the hardest things for an XSL-FO user to understand, since XSLT can match on parts of the text of a document and generate FOs for them.

Feature Comparison

- Box layout
- Page layout
- Headers & footers
- Multiple columns
- Keeps & breaks
- Paragraph setting
- Footnotes & sidenotes
- Tables
- Lists
- Character setting
- Japanese text composition
- Cross-references
- Image positioning
- MathML & SVG graphics
- Counters
- Color
- Borders & background
- PDF output
- Indexes

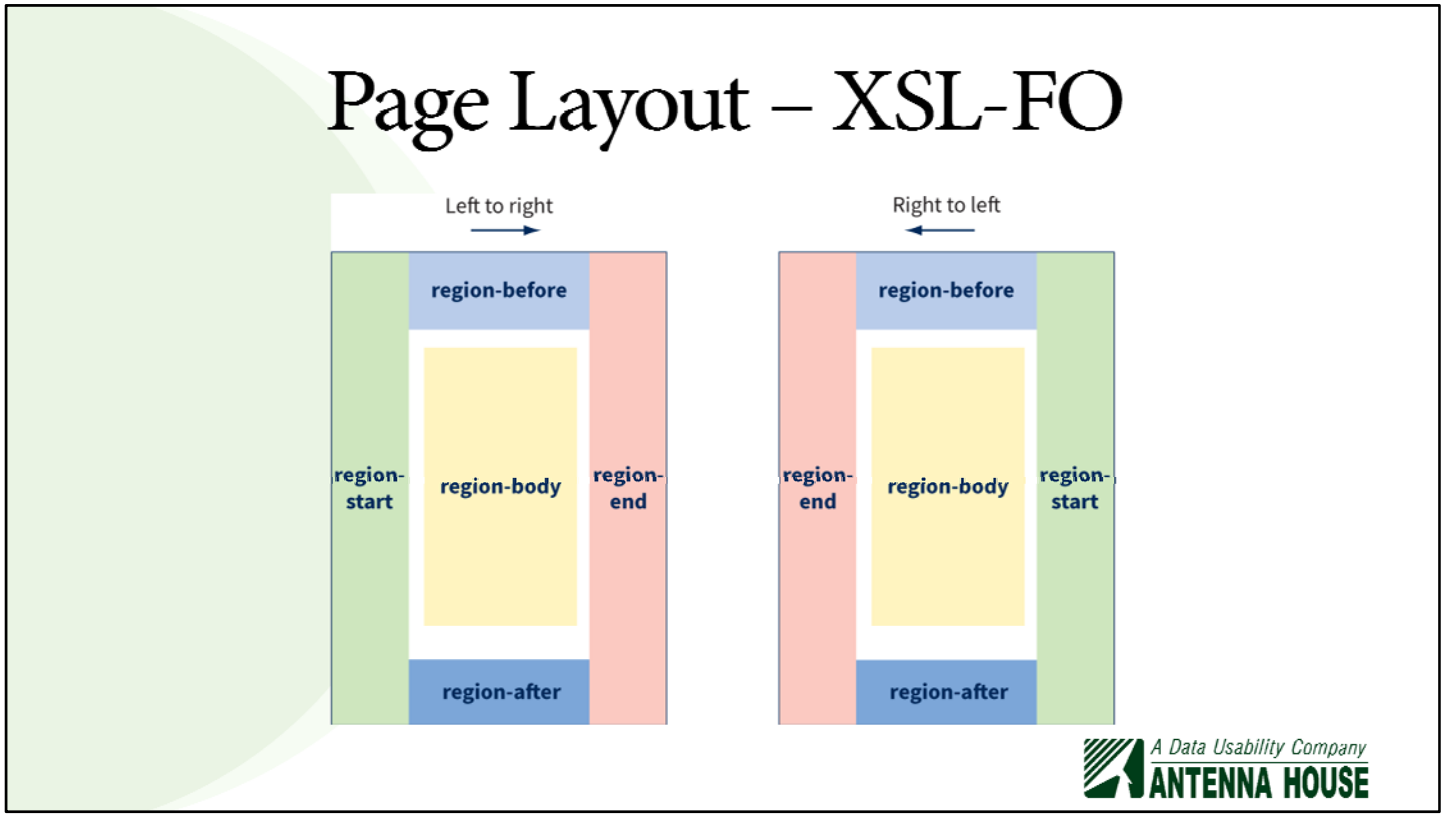
A Data Usability Company
ANTENNA HOUSE

These are the chapters of the *XSL-FO/CSS Comparison* book and the rows in the table in the paper. I'm only going to go through a few of them today…



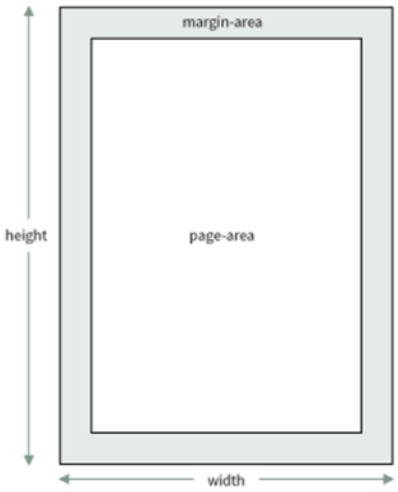
Currently, everything in both XSL-FO and CSS is made up of boxes, where the content is surrounded by padding, which is surrounded by the border, which is surrounded by the margins.

90% of what you do in XSL-FO or CSS concerns placement of boxes, and they both have the same basic properties for padding, borders, and, to some extent, margins.



For laying out pages, XSL-FO defines a simple page master that has a body region and four 'outer' regions for the headers and footers. Which outer region is which depends on the writing mode and reference orientation.

Page Layout – CSS



A Data Usability Company
ANTENNA HOUSE

CSS defines a page as having a page area surrounded by margins. As we will see shortly, the margins are divided into 16 page-margin boxes.

Headers & Footers – XSL-FO


```
<fo:static-content flow-name="Even-Header">
  <fo:block>UDHR in Unicode</fo:block>
</fo:static-content>
<fo:static-content flow-name="First-Outside">
  <fo:block-container reference-orientation="270">
    ...
  </fo:block-container>
</fo:static-content>
<fo:static-content flow-name="Odd-Outside">
  <fo:block-container reference-orientation="270">
    ...
  </fo:block-container>
</fo:static-content>
```

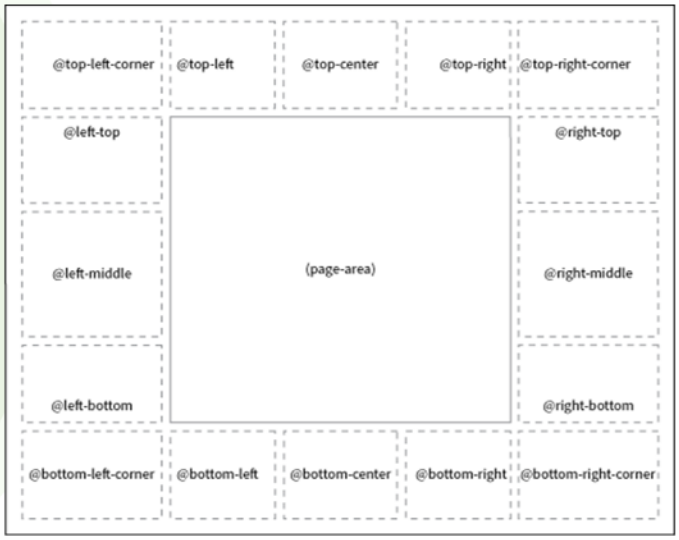
```
<fo:static-content flow-name="Odd-Header">
  <fo:block>...</fo:block>
</fo:static-content>
<fo:static-content flow-name="Even-Footer">
  <fo:block><fo:page-number/></fo:block>
</fo:static-content>
<fo:static-content flow-name="Odd-Footer">
  <fo:block><fo:page-number/></fo:block>
</fo:static-content>
<fo:flow flow-name="xsl-region-body" id="sco">
...
</fo:flow>
```

A Data Usability Company
ANTENNA HOUSE

Content for headers and footers in XSL-FO is generated as ‘static content’ that is directed to an outer region with the same name. Static content that does not have a corresponding region on a page does not appear on that page. The custom region names allows the same static content to be directed to different regions on different pages: for example, to the outer, left edge of left-hand pages and also to the outer, right edge of right-hand pages.

Static content can retrieve some or all of its content from marker FOs in the main flow of the document.

Headers & Footers – CSS

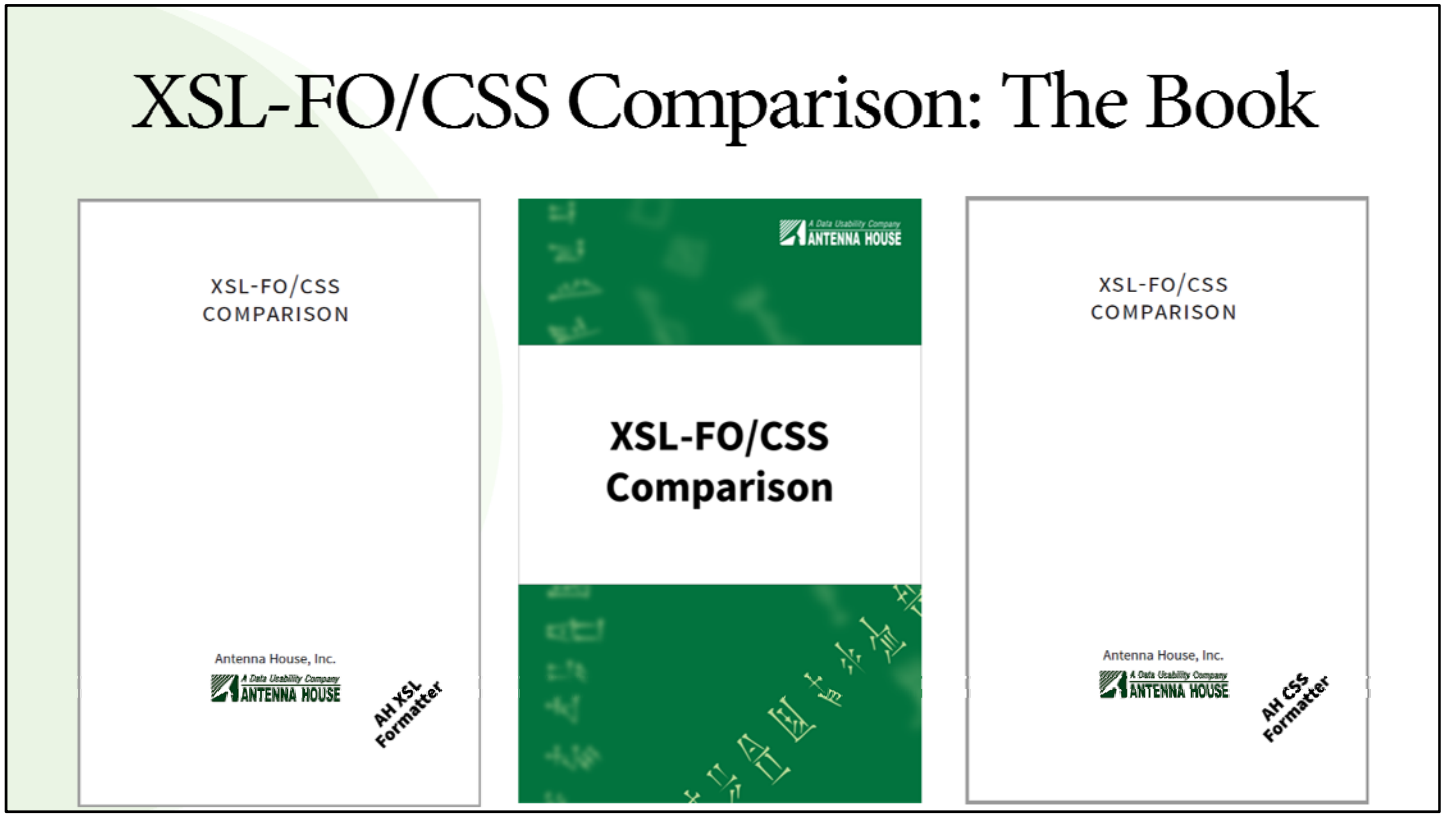


A Data Usability Company
ANTENNA HOUSE

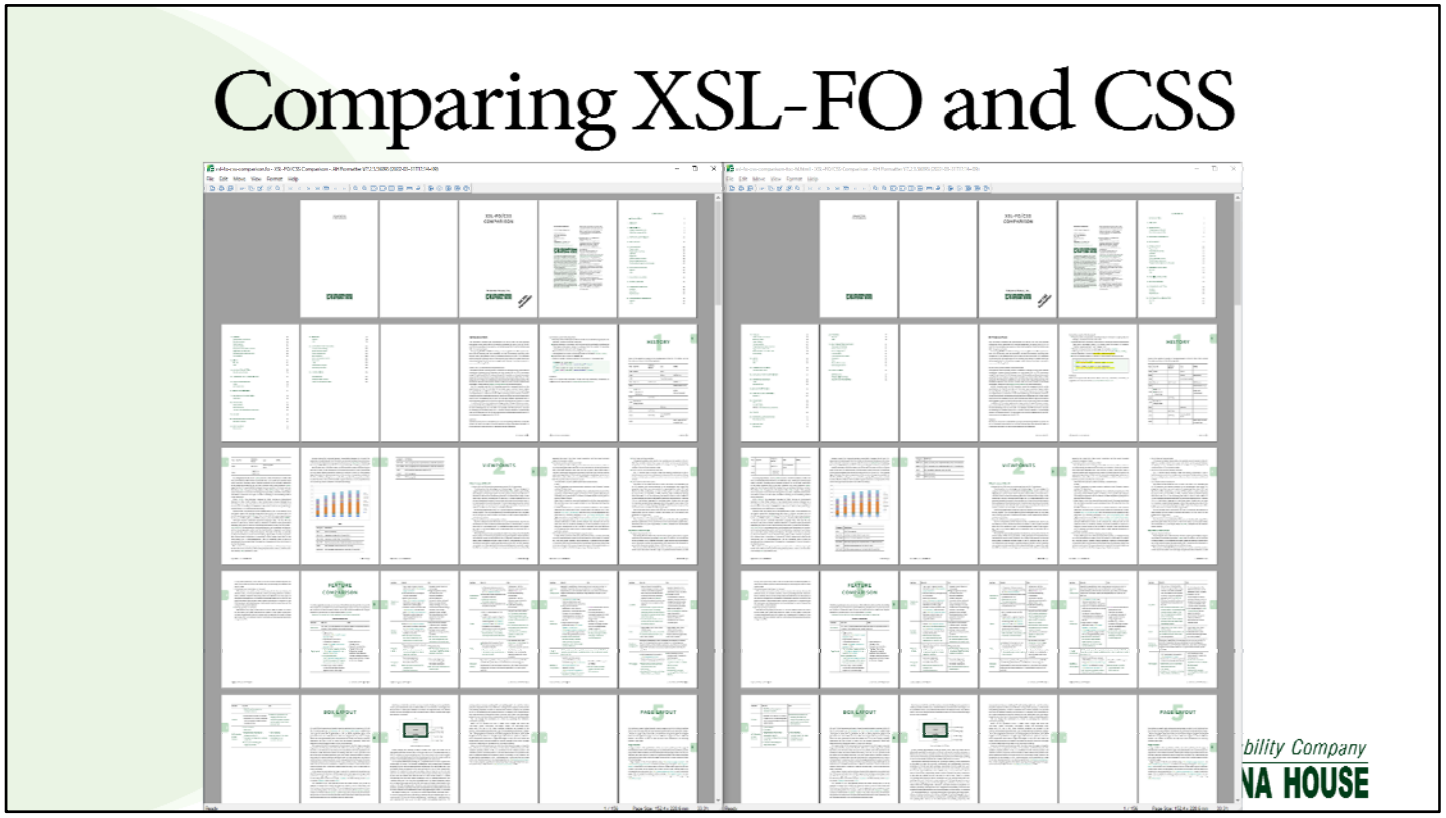
CSS defines 16 page-margin boxes on every page, where a page-margin box with content generates a box.

The content of a page-margin box can be:

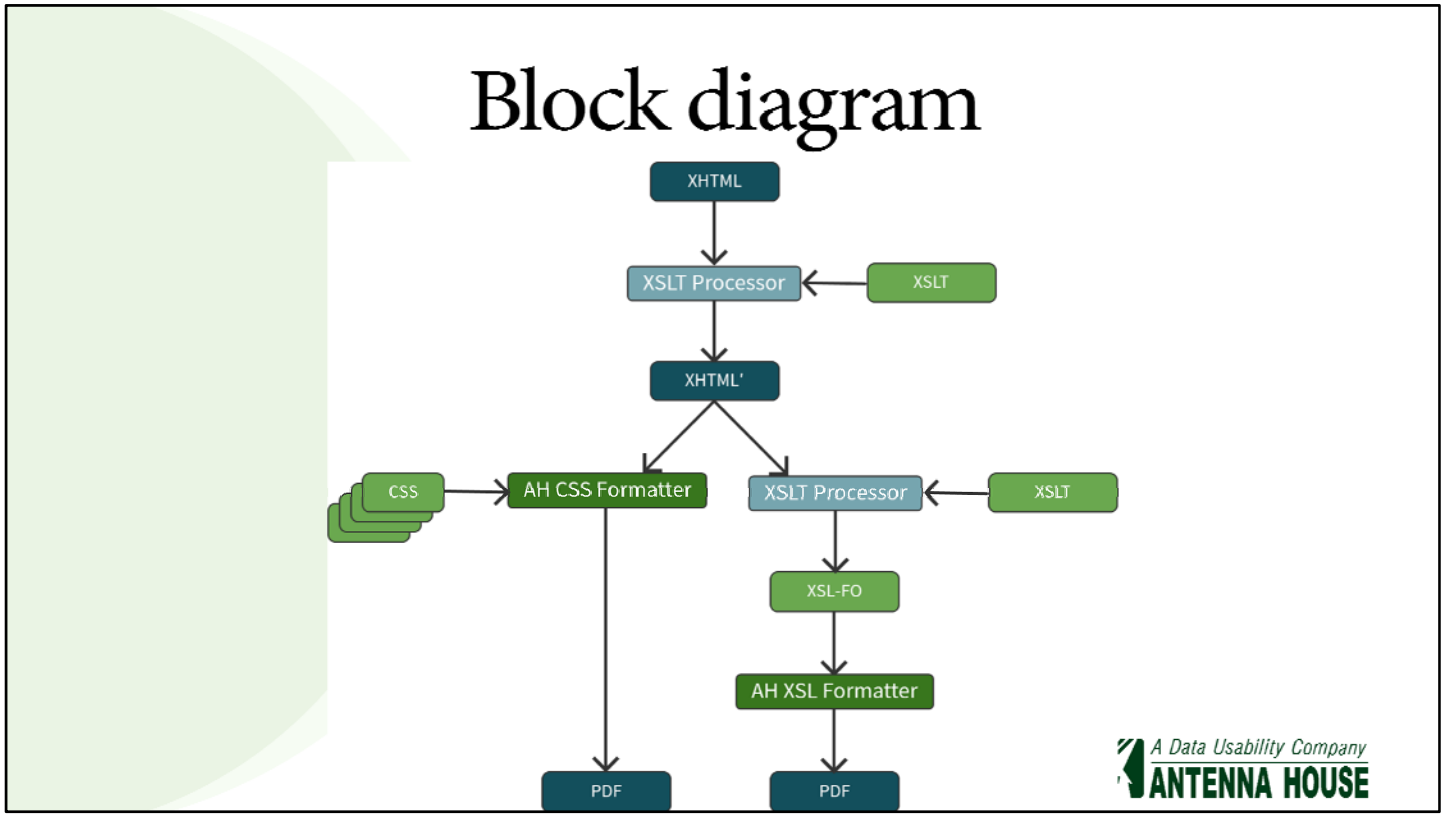
- Inline content such as: literal strings; values of named strings set by the 'string-set' property of an element; counter values; leaders; cross-reference text; or images
- A 'running' element that is taken out of the normal flow for use in page-margin boxes. Running elements are not much different from static content, except that running elements are in the main flow and not separate like static content. Repeating, say, chapter titles just for them to be removed from the flow is seldom something that you want to do by hand, which points to the need to transform a source document before formatting it with CSS.



The second half of this talk covers how we generated two versions of the book, formatted using XSL-FO and CSS, that are so identical that you need the Antenna House Regression Testing System (AHRTS) to find some of the differences.



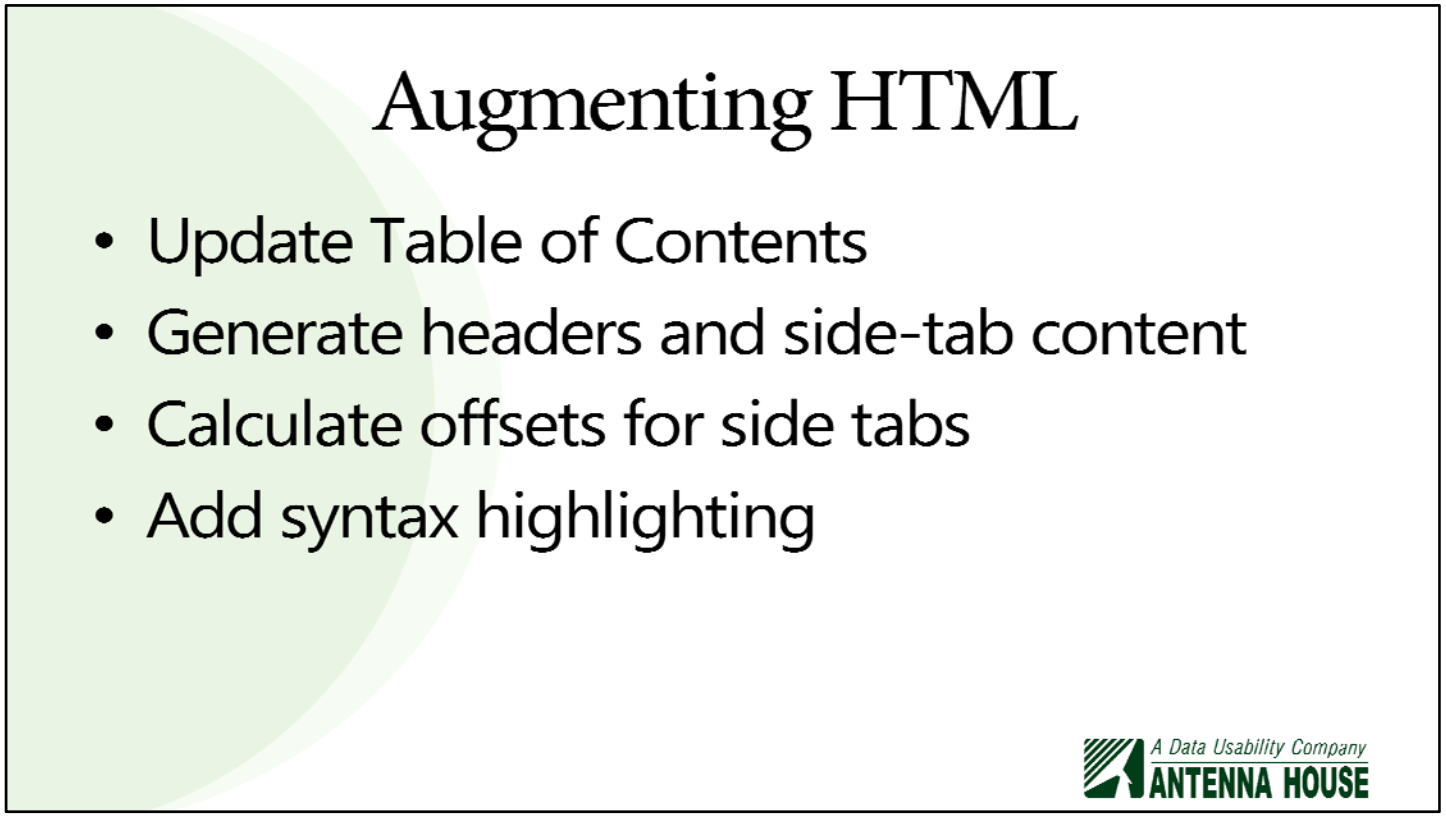
As you can see, most of the pages look identical, even down to the line breaks.



The document is written in XHTML that is then augmented to update the table of contents, generate the content for the headers, footers, and side-tabs, and even calculate the position of each side-tab.

That XHTML is formatted using AH CSS Formatter and CSS stylesheets to produce the PDF for the CSS version.

The augmented XHTML is also transformed into XSL-FO that is formatted using AH XSL Formatter to produce the PDF for the XSL-FO version.

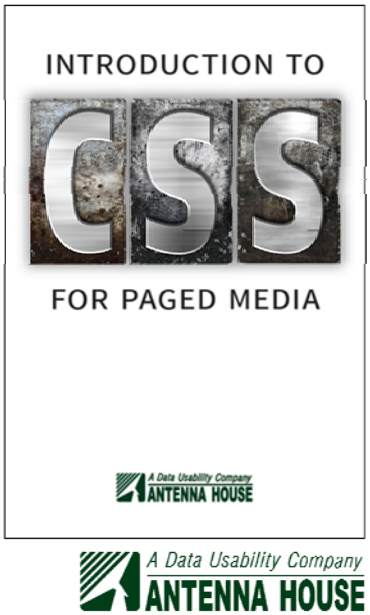


The augmented HTML contains some of the things that you either don't want to do by hand or don't want to see in the authoring version:

- Up-to-date table of contents
- Header and side-tab content
- Offset for individual side-tabs
- Markup for syntax highlighting in code samples

CSS Styles

- Based on "Introduction to CSS Paged Media" styles
- Reorganised processing for reuse but minimal style changes
- Unlikely to change much



A Data Usability Company
ANTENNA HOUSE

For faster development, and for consistency between publications, the CSS version reuses the styles from *Introduction to CSS Paged Media*.

The processing had to be reorganised so that a single set of stylesheets could be used from multiple locations, but there were minimal changes to the actual CSS.

Because the goal is to be consistent with the existing book, the CSS styles are unlikely to change much in the future.

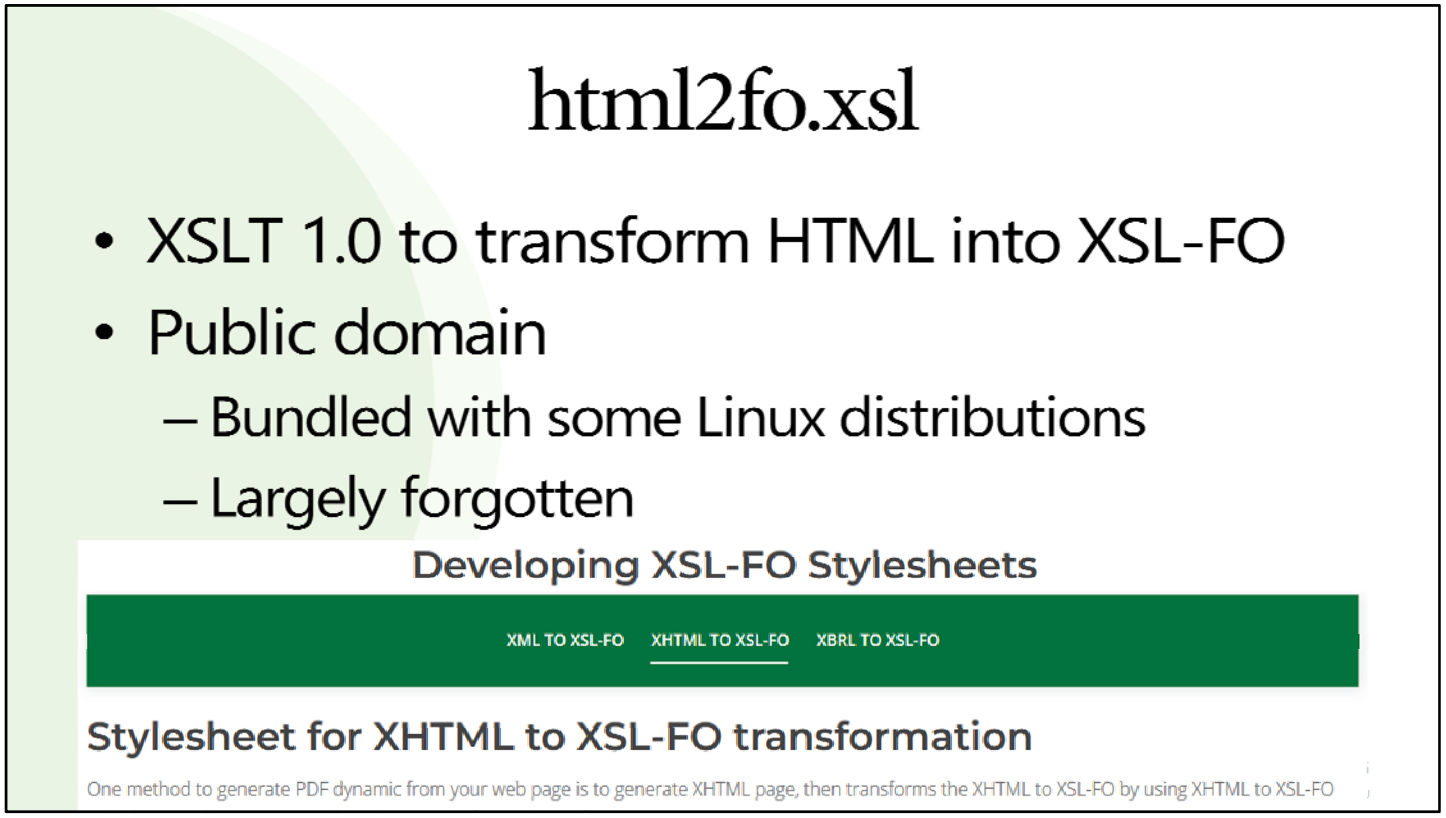
XSL-FO Styles

- Augmented HTML already includes:
  - Header and footer content
  - Side tabs
  - Syntax highlighting
- XSLT reimplements CSS styles
  - Imports XSLT 3.0 version of html2fo.xsl for core HTML-to-FO transformations

A Data Usability Company
ANTENNA HOUSE

The XSL-FO is generated from the augmented HTML because that already contains the header and footer content, side-tabs content, and syntax highlighting markup.

The XSLT implements the CSS styles on top of an XSLT 3.0 version of the core HTML-to-FO transformations that Antenna House provides on its website.



The original html2fo.xsl XSLT 1.0 stylesheet from the Antenna House website provides a basic transformation from XHTML to XSL-FO. It is designed to be imported by another stylesheet that defines a more styled appearance.

The stylesheet is public domain. It is included in some Linux distributions, but otherwise it has been largely forgotten.

xsl:attribute-set

- html2fo.xsl uses `xsl:attribute-set`
- Composes across modules
- Evaluated on every use

```
<xsl:attribute-set name="id">
  <xsl:attribute name="id"
                 select="generate-id()"/>
</xsl:attribute-set>
```

- Can't remove attributes from set

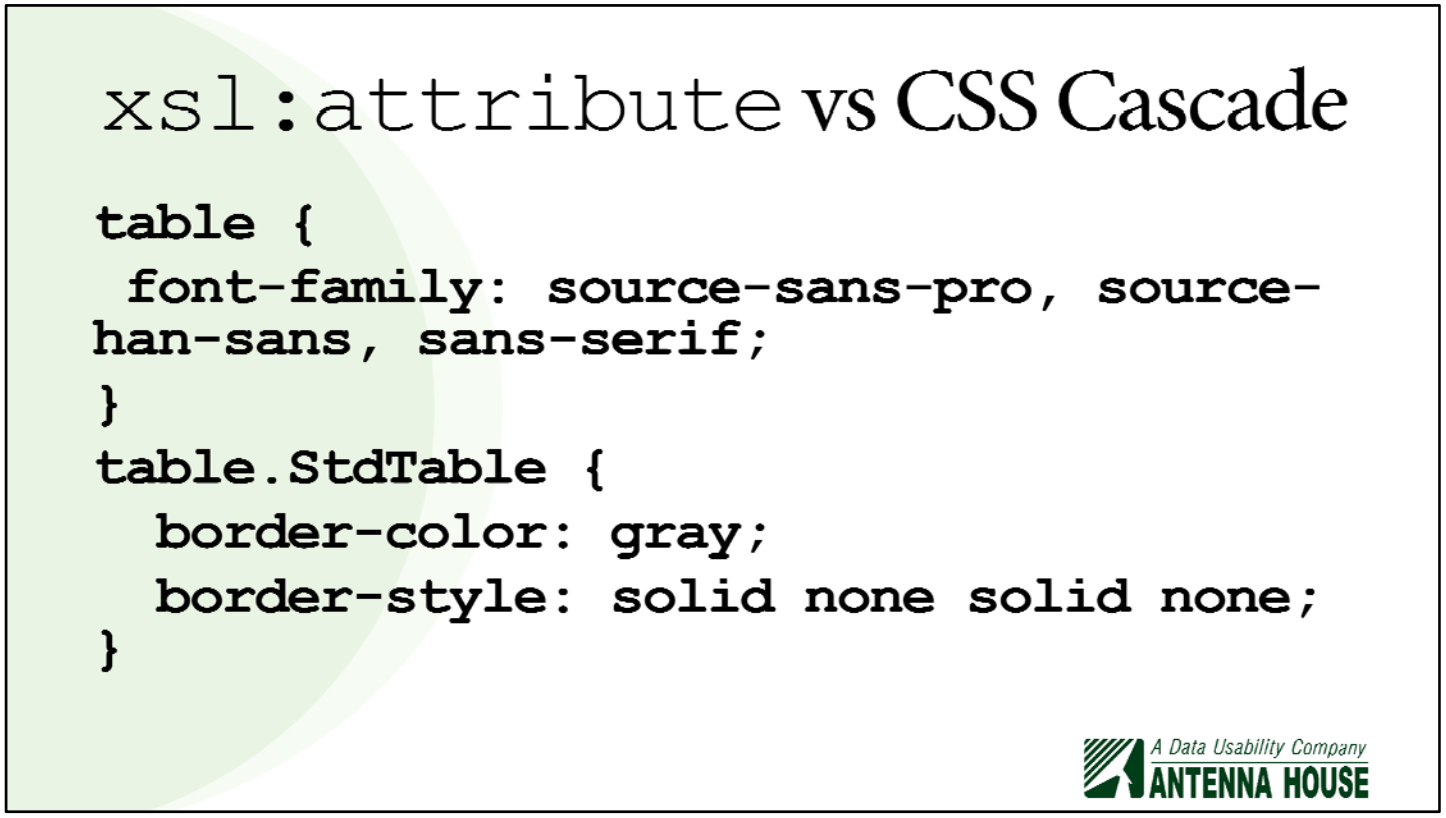
A Data Usability Company
ANTENNA HOUSE

The original XSLT 1.0 stylesheet makes extensive use of xsl:attribute-set for defining styles that can be overridden by an importing stylesheet.

xsl:attribute-set is useful because:

- Multiple xsl:attribute-set with the same name combine, with precedence rules for two xsl:attribute-set defining the same attribute
- The xsl:attribute in an xsl:attribute-set are evaluated each time they are applied, rather than the attribute's values all being the same static string. The ultimate example may be generating a unique ID for every element where an xsl:attribute-set is used.

The downside of xsl:attribute-set is that, while multiple xsl:attribute-set definitions can merge, there is no way to remove an attribute definition from an xsl:attribute-set. When generating XSL-FO, the best that can be done is to specify an attribute definition that generates either 'inherit' or the default value of a property, whichever is more useful.



xsl:attribute-set containing xsl:attribute may be seen as XSL's equivalent of how properties cascade in CSS: certainly, xsl:attribute-set is most often seen in stylesheets that generate XSL-FO.

The difference, as we know, is that CSS rules are evaluated on the fly, whereas xsl:attribute-set names are written in the stylesheet.

For this example, a 'StdTable' table uses whatever rules are defined for a 'StdTable' table or a table, and the CSS rules for specificity determine which property declarations are actually used.

In contrast, the XSLT stylesheet needs to know what attribute sets are defined that apply to a particular context. This is not so bad for a 'StdTable' table, but it gets more convoluted when defining which attribute sets to apply on a particular descendant of a 'StdTable' table.

Map of Properties (1)

```
<xsl:template match="table[tokenize(@class, '\s+') = 'StdTable']">
  <xsl:param name="table-properties"
             select="map { }"
             tunnel="yes"
             as="map(xs:string, map(xs:string, xs:string?))" />

  <xsl:variable
      name="local-table-properties"
      select="map {
'table' : map { 'border-top' : '1.5pt solid gray' }
}"
      as="map(xs:string, map(xs:string, xs:string?))" />
```

A Data Usability Company
ANTENNA HOUSE

The solution that is used is a map of maps of properties for different contexts. This differs from the other use of a map of properties in my *XSLT 3.0 Testbed* talk from XML Prague 2014.

The template for a 'StdTable' table defines the properties to apply to the <table> element (and in the real stylesheet, to its table-related descendants).

This template may also be passed a similar map of maps of properties that can override the local map.

Map of Properties (2)

```
<!-- Properties from importing stylesheet or higher-priority
     template have precedence over local properties. -->
<xsl:variable
    name="table-properties"
    select="map:put($table-properties,
                    'table',
                    map:merge(($table-properties('table'),
                               $local-table-properties('table'))))"
    as="map(xs:string, map(xs:string, xs:string?))" />
```

A Data Usability Company
ANTENNA HOUSE

The overriding properties are merged with the local properties, with the overriding properties taking precedence.

Map of Properties (3)

```
    <xsl:next-match>
      <xsl:with-param
          name="table-properties"
          select="$table-properties"
          tunnel="yes"
          as="map(xs:string, map(xs:string, xs:string?))" />
    </xsl:next-match>
  </xsl:template>
```

A Data Usability Company
ANTENNA HOUSE

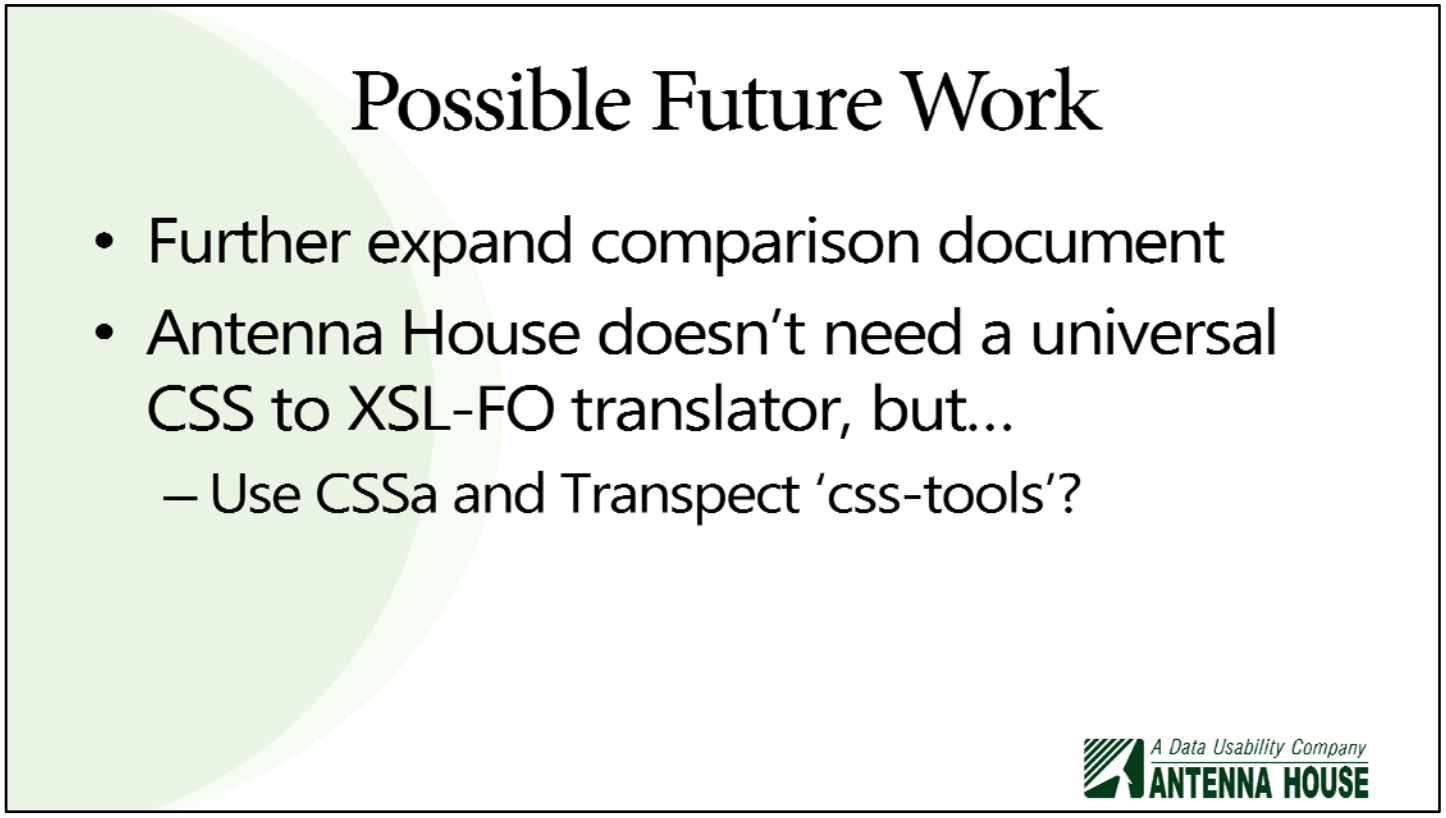
The resolved properties are passed to the ‘html2fo.xsl’ templates for tables.

Map of Properties (4)

```
<xsl:template name="process-table">
  <xsl:param name="table-properties"
             select="map { }"
             tunnel="yes"
             as="map(xs:string, map(xs:string, xs:string?))" />
<xsl:sequence
      select="ahf:add-properties($table-properties('table'),
                                 ($border-properties,
                                  'border-collapse'))" />
  <xsl:apply-templates select="col | colgroup"/>
  <xsl:apply-templates select="thead"/>
  <xsl:apply-templates select="tfoot"/>
  <xsl:apply-templates select="tbody"/>
</xsl:template>
```

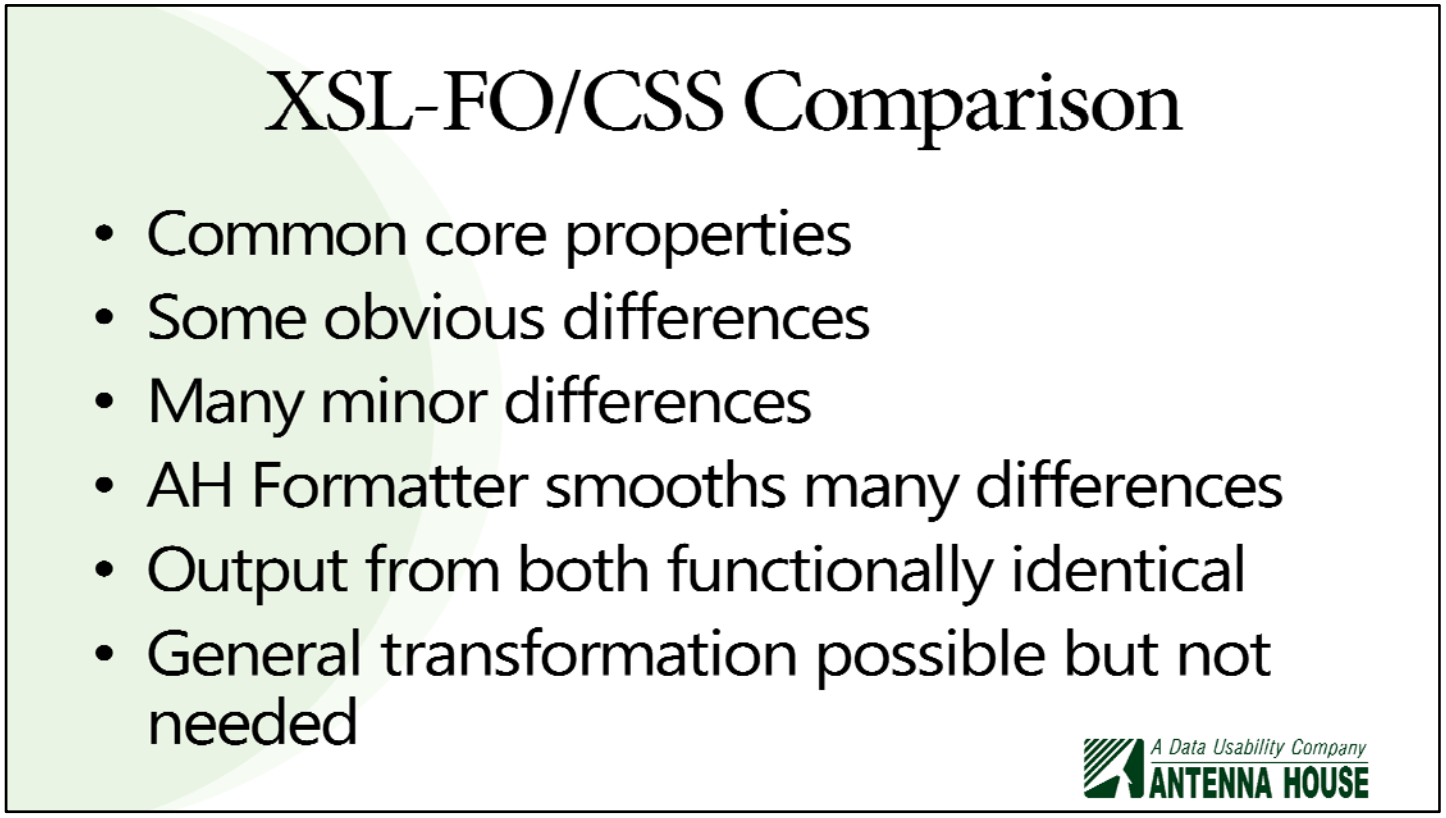
A Data Usability Company
ANTENNA HOUSE

The 'html2fo.xsl' template generates attributes for the entries in the applicable map. Because a single HTML element can generate more than one FO, it is possible to select which properties from the map to apply to each element.



Possible future work includes:

- Expanding the comparison document. Even though the comparison document has expanded from a table into a 150-page book, the closer you look, the more details there are to find.
- Antenna House has a CSS formatter, so it does not need a general-purpose CSS to XSL-FO translator. A general-purpose translator would also need to handle the effect of 'style' attributes on individual elements, which is not needed for this document. However, a general-purpose CSS to XSL-FO translator could be made by using the Transpect 'css-tools' utility to resolve all applicable CSS properties into added CSSa attributes then transforming the annotated HTML into XSL-FO. Because the translation is done in XSLT, such a system could even handle extensions such as Oxygen's 'oxy_xpath()'.



In this talk, we have seen that XSL-FO and CSS have a common core of identical properties. While there are some obvious differences between XSL-FO and CSS, such as how page masters are selected and content is directed to the headers and footers, there are also many minor differences, enough to make a 150-page book.

However, AH Formatter smooths out many of the differences between XSL-FO and CSS, so much so that it's possible to format the same content with both XSL-FO and CSS and generate pages that are functionality identical.

Antenna House has a CSS formatter, so it does not need a general-purpose CSS to XSL-FO translator. Indeed, this project tried to invent as little as possible: the CSS styles were reused from a previous book, and the transformation to XSL-FO reused the existing 'html2fo.xsl' stylesheet. However, a general-purpose translator could have been developed if it had been needed.

Thank you for your time.

The *XSL-FO/CSS Comparison* book and the HTML+CSS and XSL-FO sources are available from http://www.antennahouse.com/xsl-fo-css-comparison.

[End]