

# Web-Based Presentations

Erik Wilde (UC Berkeley)  
Philippe Cattin, Felix Michel (ETH Zürich)

## Abstract:

The management and publishing of complex presentations is poorly supported by available presentation software. This makes it hard to publish usable and accessible presentation material, and to reuse that material for continuously evolving events. XSLidy provides an XSLT-based approach to generate presentations out of a mix of general-purpose HTML and a small number of presentation-specific structural elements. Using XSLidy, the management and reuse of complex presentations becomes easier, and the results are more user-friendly in terms of usability and accessibility.

## 1 Introduction

Web content most often is produced to be read online or to be printed on paper, but there are also many settings where content is intended to be presented as auxiliary material for an aural presentation. Typical scenarios for this kind of setting are business presentations, courses, and lectures. There is a wide spectrum of different requirements for these scenarios, which in most cases depend on the amount of presentation material that is prepared, the structure of this material, the need for structured presentations, and the requirement to make the presentation material available in some reusable form to the audience.

When we speak of “complex presentations” in the following text, we refer to different aspects which in our practical experiences are typical for larger presentations. These aspects include

- larger volumes of presentation material than a small number of slides, usually intended for multi-day presentations or semester-long university courses,
- the ability to use structuring methods which group these contents into units, very much like chapters, sections, and sub-sections in books,
- support for using these structures for presentation purposes, such as generating table of contents, or creating cross-references to other parts of a presentation, and finally
- general support for content which is Web-style content, using standard Web content mechanisms such as links, and introducing useful shortcuts for common use cases, such as the ability to manage all images for a presentation in a dedicated directory.

Today, presentation material is often prepared using commercial software packages and then “published” on the Web as PDF. Presentation software excels at providing visual effects, but provides little support for handling structured and complex presentations. Furthermore, because of the very nature of dedicated software and proprietary file formats, presentations prepared with these packages are not very usable and accessible for anybody not using that particular software.

The two most popular software packages today used for presentation are *PowerPoint* and *Keynote*. Both provide a large variety of templates for presentations as well as visual effects for slide transitions and incremental display of slides. While the result of these visual effects are something that in most cases is more distracting than helping to convey the message of the presentation [BC03], it is more important to notice that both software packages provide virtually no support for structured presentations. Especially for more complex presentation material (such as university courses), this lack of support makes it rather hard to create and maintain complex presentations with these software packages. In addition, even though there is support for HTML [RLJ99] export in both packages, the results are far from being an appropriate Web representation of the original content of the presentation.

The important difference between mere *exporting for presentation on the Web* and *representing a presentation as Web content* is that exported presentation contents often fail to be usable and accessible Web content. Some packages export presentations as images or even videos, which makes contents invisible to search engines and make navigation of the contents as a hypermedia document impossible. Instead, Web content should preserve the full contents of a presentation as navigable Web structures, ideally even adding additional navigation facilities for improving accessibility. Furthermore, content should be usable for online viewing or printing, supporting as many uses of the content as possible.

Broadly speaking, the subject of this paper can be seen as part of the e-learning and open content idea, with the important distinction being that our approach addresses presentation material exclusively, and not a complete set of tools and technologies for managing e-learning content. Some of the challenges of the broader e-learning and open content ideas are cultural [VHD06], others are technical [BBJJ04], and our work contributes to the technical area by presenting a way of how to publish and manage open content. The ability to manipulate lecture and presentation materials at a more granular level can facilitate collaborative forms of development of e-learning resources. Remixes, revisions, and new applications of course content can result more easily.

In this paper, we describe a solution that has been developed because of a real-world need. For university courses, it would be ideal to have Web-based presentations which not only are accessible and can be viewed online or printed, but which also allow the creator of the presentation to turn the course material into a sophisticated hypermedia resource augmenting the course. It should be able to have structuring on different levels, use cross-references between slides and parts, manage additional material augmenting the course (such as images and listings and other resources), and generally manage the presentation material in a way which guarantees a consistent and easily updatable representation of the course material on the Web.

While the support for complex presentations is something where only little work has been done so far, there is some prior work regarding Web-based presentations. The two most notable attempts to produce HTML-based presentation so far are *HTML Slidy* [Rag06] and *S5*<sup>1</sup>, both use a combination of JavaScript scripting and CSS stylesheets to create Web pages which can be used as presentation material in standard browsers. The advantage of HTML-based presentations is that they can be used with any state-of-the-art Web browser on any platform, and thus provide a high level of accessibility. The disadvantage of both

---

<sup>1</sup><http://meyerweb.com/eric/tools/s5/>

approaches is that the presentations have to be produced manually, in both cases with a mix of mostly HTML, and a bit of package-specific code (in both cases in the form of specific `classes` for HTML elements) which then binds the scripting and styles to the presentation.

Since the goal is to support more advanced features such as dynamic outlines, table of contents, cross-references, transclusion, and the ability to add generated content, we decided to base our work on the Slidy package (which provides a solid foundation for Web-based presentation), but to use an additional transformation step, which takes the input documents, uses XSLT 2.0 [Kay07] to transform it into a number of result documents (such as individual Slidy documents and table of contents for inclusion in a Web page), and thus can implement functionality which is impossible to support without preprocessing. Because our package's two main foundations are XSLT and Slidy, we have named it *XSLidy*.

The remainder of this paper is structured as follows: We first describe the implementation of XSLidy (Section 2), as well as the major features of the package. The most important feature of XSLidy is its support for structured presentation, which is described in Section 3. In addition to these linear, traditional document-style structures, XSLidy also supports advanced hypermedia features, as described in Section 4, so that the resulting set of Slidy documents can be regarded as one hypermedia document. Styling (Section 5) is mostly done using CSS and is very flexible, so that layouts can be adjusted to fit personal preferences and/or corporate design guidelines. Finally, there is an extension mechanism (Section 6) which allows users to add functionality to XSLidy, and Section 7 explains this mechanism using an existing extension for rendering mathematical formulas as an example.

## 2 Implementation

Since the presentation material should be HTML-based, XSLidy documents are using HTML as the foundation for their markup. However, XSLidy introduces a small number of additional elements,<sup>2</sup> which provide support for functionality not available in HTML, such as structuring on various levels, or textual inclusion. As shown in Figure 1, XSLidy is implemented in XSLT 2.0 [Kay07], and takes as input the presentation document as created by the user. Since we use XSLT, the document has to be a well-formed XML [BPSM<sup>+</sup>06] document, and thus instead of HTML, XSLidy requires XML-compliant XHTML [Pem02] as its basic vocabulary.

Most of the HTML elements are not processed at all by the code, but the specific XSLidy elements are transformed into HTML which then becomes part of the individual documents, which basically are Slidy presentations (with some additional features added by XSLidy to extend Slidy's functionality). For example, the structuring of the content that can be inferred by analyzing the structure of the input document is used to generate outline slides (an example is shown in Figure 4a), which are used between individual parts of a presentation to give a visual representation of the various parts that a presentation might have. This feature, as many others, of course can be controlled by the user, which enables XSLidy to produce presentations in a very flexible way.

---

<sup>2</sup>To simplify authoring, we accept XSLidy and XHTML elements to be either in the XSLidy namespace or in the HTML namespace.

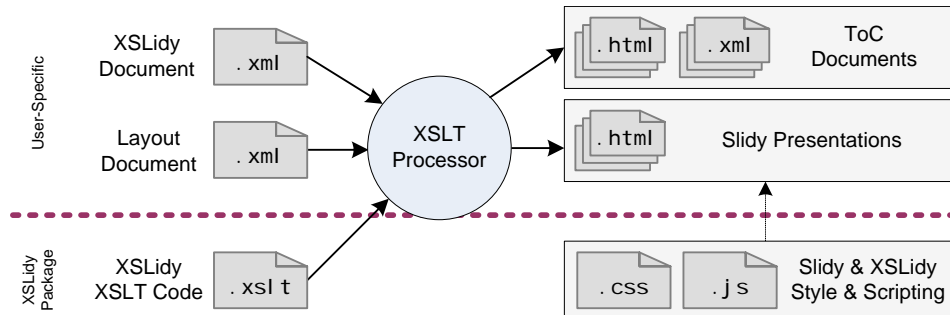


Figure 1: XSLidy Processing and Results

In addition, it is possible to generate *Table of Contents (ToC)* documents, which usually provide a summary of the presentation for use in some context other than Slidy-based presentation. Examples for this are ToC documents in HTML, which are used as an overview of the complete presentation (an example is shown in Figure 4b), or XML summaries of the presentation, which may be required as input for further processing in an XML-based publishing chain.

The major part of XSLidy is the transformation of the XML input document into the individual result documents as described above. However, XSLidy also adds some runtime functionality to the scripting provided by the original Slidy package. These additional functionalities include presenter-supporting features which are commonly found in convenient presentation software. For example, a stop watch indicates for how long the current slide has been on display, and for how long the current presentation has been running so far.

A more interesting enhancement of the scripting code provides for *distributed* or *remote presentations*. The creator of Slidy, DAVE RAGGETT, proposes [Rag06] to use the *Real-Time Streaming Protocol (RTP)* together with HTTP (through the `XMLHttpRequest` facility) to create distributed presentations. As the processing and streaming of audio data cannot be done without additional browser plugins to be installed, we decided not to address this part of the problem. Instead, we only use HTTP requests, thus preserving the advantages of plugin independence and cross-browser and cross-platform operation. Furthermore, the audio part of remote presentations can easily be taken care of by free and high-quality Internet telephony applications such as Skype<sup>3</sup> or similar products.

As shown in Figure 2, remote presentations require a tiny PHP application to be installed on the server that hosts the presentation. This server-side application maintains the state information of each remote presentation session. Sessions can be created by any client (which then becomes the leader of this session), and other clients can subsequently join one of those sessions. Once a client has joined a session, the presentation slides are displayed in congruence with the slides that the session leader displays.

Synchronous slide changes are implemented by client-side HTTP polling. In order to limit bandwidth consumption, the server makes use of the HTTP response code 304,

<sup>3</sup><http://www.skype.com/>

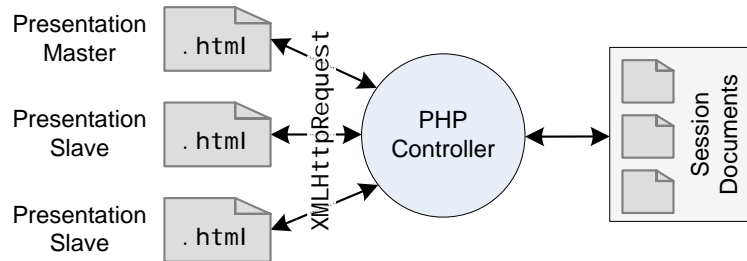


Figure 2: XSLidy Remote Presentations

which indicates that the requested resource has not been modified. Although this approach introduces a maximum latency of one polling interval (and although it lacks perfect elegance), the chosen approach relieves the server-side application from the burden of keeping track of all clients, and it avoids problems with firewalls and *Network Address Translation (NAT)*, because all connections are HTTP connections using port number 80, and all connections are initiated from the clients.

### 3 Structuring

The most apparent lack of support for complex presentations of today’s presentation packages is that they fail to support the structuring capabilities (such as individual presentations and sections and subsections within these) which are required when creating complex content structures. As a result, users have to resort to makeshift ways of capturing this structure; these brittle structures often fail to adapt properly to the reuse of material in different contexts, and the continuous modification of the material for different versions of the presentation.

In contrast, XSLidy represents presentation structures as XML, as shown in Figure 3, and it manages complete presentations (such as a whole university course) as one XML document.<sup>4</sup> Most of the elements in this listing are from the XSLidy vocabulary, but this is only the case because the figure presents the parts of the source which focus on structural information. Usually, most of the elements in an XSLidy presentation are HTML elements.<sup>5</sup>

While there is an *XML Schema* [TBMM04] for XSLidy, it is structurally not very interesting, because of the fact the actual XSLidy documents contain a mix of XSLidy and HTML elements. Furthermore, XSLidy’s relaxed approach towards namespaces cannot be encoded in the schema. Thus, the schema serves more as part of the documentation, and not so much as an input to possible validation of XSLidy documents. It would probably make more sense to describe the rules of how to use XSLidy elements in a rule-based

<sup>4</sup>If required, users can use inclusion features which provide access to various physical documents, but the logical document which is processed by the XSLidy code is one XML document. The inclusion process uses an XSLT implementation of *XML Inclusions (XInclude)* [MOV06] called *XInclude Processor (XIPr)* [Wil07].

<sup>5</sup>For example, the XSLidy XML source for <http://dret.net/lectures/xml-fall107/>, a university course managed in XSLidy, contains 7554 elements, of which 5768 (76%) are HTML elements, and 1786 (24%) are XSLidy elements.

```

1  <xslidy xmlns="http://dret.net/xmlns/xslidy/1" xmlns:xslidy="http://dret.net/xmlns/xslidy/1">
2  <title short="XML Foundations"><a href=".">XML Foundations</a> (INFO 242)</title>
3  <author short="E. Wilde"><a href="http://dret.net/netdret/" title="dret.net">Erik Wilde</a></author>
4  <affiliation short="UC Berkeley iSchool"><a href="http://www.berkeley.edu/" title="University of California, Berkele
5  <date short="Fall 2007">Fall Semester 2007</date>
6  <copyright>2007 Erik Wilde</copyright>
7  <style type="text/css" src="xslidy-fall07.css"/>
8  <toc id="html-toc" name="toc.html">
30 <toc id="sylvia" name="242.xml">
157 <presentation id="intro" cover="slidycover">
727 <presentation id="example" cover="slidycover">
737 <presentation id="basics" cover="slidycover">
738 <title short="Basics">XML Basics</title>
739 <date>2007-09-04</date>
740 <toc id="resources"><a href="http://www.w3.org/TR/REC-xml/" title="W3C XML 1.0 Specification">Spec</a></t
741 <toc id="abstract">The <em>Extensible Markup Language (XML)</em> defines a simple way for structuring da
742 <slide>
743 <title>Abstract</title>
744 <p class="abstract"><toc id="abstract"/></p>
745 </slide>
746 <part>
747 <title>Foundations for XML</title>
748 <slide>
749 <title>Identifications</title>
750 <ul>
751 <li>Identification of Character Encodings</li>
752 <ul>
753 <li>text can be encoded using different character sets and encodings</li>
754 <li>IANA maintains the <a href="http://www.iana.org/assignments/character-sets">official list of char
755 </ul>

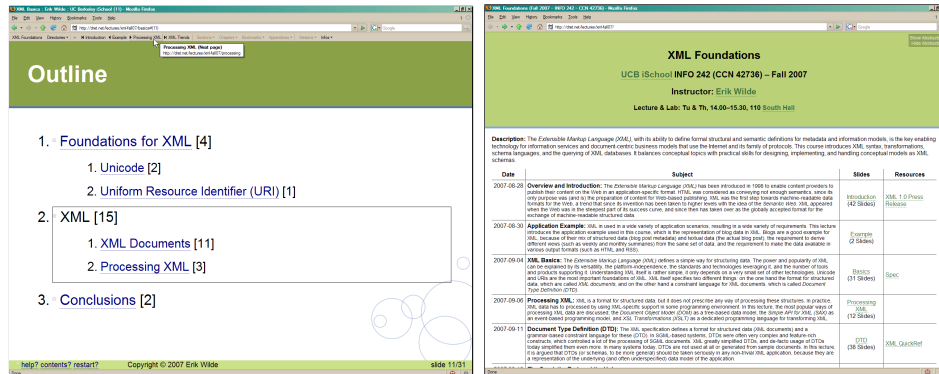
```

Figure 3: XML Structures Representing Presentation Structure and Content

schema language such as *Schematron* [Int06], which would be much better suited to validate real-world XSLidy documents.

Lines 1 through 7 contain information which pertains to the presentation as a whole, in this case a university course about XML foundations. The presentation uses a custom style (line 7), and two ToC documents are produced for it, the first is used as an overview in the courses Web page (which is shown in Figure 4b), whereas the second is used as input to a syllabus viewing application which requires XML documents for collecting syllabus information across various courses. Both `toc` elements contain templates which are then filled in with the details found in individual `presentation` elements. These `presentation` elements (lines 157, 727, and 737) are the containers for the material for one individual representation, in this case one lecture. For each lecture, as shown on lines 738-741, some details are provided which are used for generating the Slidy code for the presentation, as well as for the ToC documents.

Each presentation uses `part` and `slide` elements for presentation structures and individual slides. It is also possible to reuse information from the presentation that is intended to be used in ToC documents. For example, line 744 uses the text of the lecture's abstract, which is given on line 741 (and used for the HTML as well as for the syllabus XML documents), and reuses it as the body of a slide, so that the slide directly following this presentation's title slide contains the abstract. A presentation's `parts` are used to generate structured presentations, as shown in Figure 4a, where the outline of a presentation, as inferred from the XML structure, has been transformed into a nested HTML list.



(a) Outline Slide

(b) Table of Contents

Figure 4: HTML Content Generated from XSLidy Structural Information

In addition, XSLidy also attempts to make as much structural information as possible available through standard Web mechanisms, for example using HTML's document relationships which are represented by `link` elements. The navigation toolbar in Figure 4a shows how, given a browser supporting this feature, the document relationships can be used to easily navigate the various HTML documents which make up the complete presentation. This feature is one example how XSLidy attempts to produce HTML which not only is suitable as presentation material, but also is rich hypermedia content, thus supporting the use of the presentation material as regular Web documents.

## 4 Hypermedia

As described in the previous section, XSLidy not only supports structured presentations, it also turns these structures into hypermedia which make it easier to navigate presentations. All links shown in Figure 4 are generated as a result of turning document structures into navigable hypermedia structures. In addition to that, users are supported in various ways to create richly interlinked hypermedia more easily.

As one example, with the help of the XSLT processing step, XSLidy supports hyperlinks on all elements (as suggested for XHTML 2.0), so that any element which should be a link simply gets an additional `href` attribute. In addition to being more succinct and semantically appropriate than using nested elements (i.e., nesting the element in question into an `a` element), this also allows for the dedicated styling of these links, image links (which in XSLidy are created most easily by using ``) can then be styled using a CSS `a .img` selector.<sup>6</sup> Additionally, links are classified whether they are intra-presentation or external links, and can then be visualized differently, too, if required.

One frequent property of complex presentations is that they use additional resources,

<sup>6</sup>XSLidy adds the element name as the `class` of a link when links are generated using the `*/@href` mechanism.

such as images or examples. HTML does not support the concept of *transclusion* (the seamless integration of resources from various locations, without losing their original context). XSLidy provides as much transclusion support as possible, for example text transclusion: the transcluded fragment is included in a presentation, and its text is turned into a link to the source. This makes it very easy to create and maintain examples in presentation material, which has not to be physically included in the presentation material, but can be managed and updated in a dedicated directory, for examples for all listings with example code. When the example code is updated, it will be transcluded when the presentation material is refreshed, and users of the presentation material can simply download the examples by clicking on them, because they are links to the example source document.

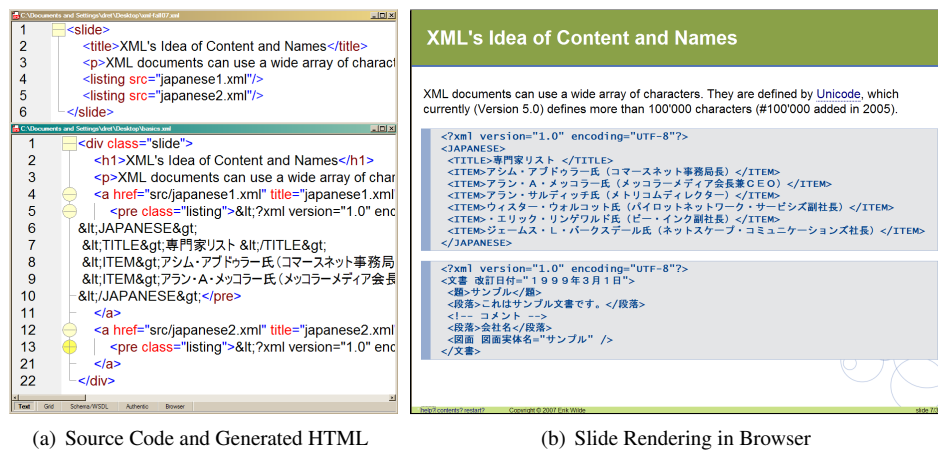


Figure 5: Text Transclusion

Figure 5a shows the source code and the resulting HTML for text transclusion. The `listing` element specifies the document to be transcluded (the base for this relative URI can be configured). The presentation contains the document's contents and is shown in Figure 5b, where special formatting is used for transcluded content, and the content is turned into a link (as shown on lines 4 and 12 of the resulting HTML), so that users can simply click on the listing to download the listing's source document.<sup>7</sup> Transclusion is one example where XSLidy attempts to preserve as many semantics as possible in the HTML (in this case using `class="listing"` to classify listings), so that styling can be based on content semantics as easily as possible.

<sup>7</sup>XSLidy supports image transclusion as well, by turning an image which is not a link into a link pointing to the image source. This is useful because images are often scaled in presentations to fit the presentation medium, and current browsers implement poor anti-aliasing methods, yielding low-quality rendering of images. Being able to look at an image in its original size therefore is a useful feature.

## 5 Styling

Since Slidy and XSLidy are Web-based presentation tools designed for standards-compliant browsers, as much styling as possible is done using *Cascading Stylesheets (CSS)* [BcHL07]. For some tasks, however (such as the transclusion functionality described above), CSS does not provide support for this kind of functionality, so they are either implemented in the initial XSLT transformation, as runtime scripting in JavaScript, or as a combination of these two methods.

The problem of how to represent content for on-screen presentation as a pageable Web page for presentation use, as a regularly scrollable Web page, or for printing as a scrollable presentation with different properties than the on-screen version, can be solved entirely in CSS. Slidy already comes with CSS code that supports on-screen viewing in two modes (scrollable and pageable) using CSS's *media types*, and XSLidy's CSS takes care of the fact that some of the XSLidy-generated content for on-screen presentation (such as the outline slide shown in Figure 4a) is not suitable for printing.

## 6 Extensions

XSLidy does not provide support for specific application areas, it is intended as a general-purpose package for anybody who wants to create Web-based presentations and has to maintain complex presentation material. However, in many cases special application scenarios require support for additional functionality, which would not be appropriate in a general-purpose package, but should be easy to integrate into such a package.

Thus, an extension mechanism is provided which can be used to plug extensions into XSLidy. Extensions can plug into XSLidy's XSLT, and/or can have additional pre- and/or post-processing stages. XSLidy's XSLT is written in a way so that extensions can override templates or add new ones for adding functionality. Pre- or post-processing stages need some kind of scripting language, and XSLidy itself does not need scripting, it is using XSLT only. However, extensions needing scripting should be written in a cross-platform scripting language, a typical example for such a language is Perl. Section 7 describes a description which introduces new elements and processing into XSLidy's XSLT, also also adds some Perl processing for additional functionality.

## 7 Formu $\LaTeX$

In many presentations for scientific disciplines, it is necessary to include mathematical formulas in the content. HTML has no support for mathematical formulas (apart from the small repertoire of characters available through Unicode), and even though there is W3C's *Mathematical Markup Language (MathML)* [ABC<sup>+</sup>03], support for this language in browsers is virtually non-existent. In addition, many scientists have their mathematical formulas readily available as  $\LaTeX$  [Lam94] code, so an easy way to have formulas in presentations is to use this code.

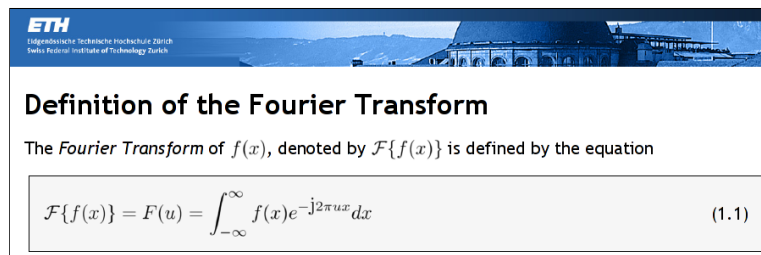
The Formu $\LaTeX$  extension allows for seamlessly integrating  $\LaTeX$  code into presentations, an example for this is shown in Figure 6. Formu $\LaTeX$  consists of XSLT code that processes `tex` elements, produces individual files containing the  $\LaTeX$  code for each formula, and a Perl script that subsequently converts these  $\LaTeX$  snippets into *Graphics Interchange Format (GIF)* or *Portable Network Graphics (PNG)* images (based on user

preference) using a transparent background. Scale and baseline information for each formula is stored in a separate file. These parameters are required to correctly position and scale the formulas within the HTML text flow. Since all sizes in the generated HTML are given in `ex` units, the scale and baseline settings are invariant to dynamic font size changes during a presentation.

|   |
|---|
| The following code shows how FormulaTeX is used:                                    |
| <pre>&lt;tex&gt;\$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}\$&lt;/tex&gt;</pre>              |
| This code displays the quadratic equation: $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ |

Figure 6: Inline Formula Rendering with FormulaTeX

To further ease the typesetting of equations, the `tex` element supports some attributes (`<tex pkg="..." id="..." class="...">`). The `pkg` attribute contains a list of comma separated LaTeX packages which need to be included to properly render the equation in LaTeX. If an `id` attribute is specified, XSLidy automatically assigns a unique equation number. The equation can then be referenced from anywhere within the presentation by using the pre-defined equation counter `EQ` with `<counter name="EQ" ref="..." />`, where `ref` has to match the equation `id`. The `class` attribute can be used to style specific equations with for example a surrounding box. Figure 7 shows a screenshot of a representative example using all these attributes. The example also shows that LaTeX equations can not only be used as a display equation, but they can be also integrated into running text and are scaled to its size.



**ETH**  
Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

## Definition of the Fourier Transform

The *Fourier Transform* of  $f(x)$ , denoted by  $\mathcal{F}\{f(x)\}$  is defined by the equation

$$\mathcal{F}\{f(x)\} = F(u) = \int_{-\infty}^{\infty} f(x)e^{-j2\pi ux} dx \quad (1.1)$$

Figure 7: Display Formula Rendering with FormulaTeX

To save network bandwidth and to reduce the processing load when rendering the LaTeX images, two strategies are employed. On the one hand, multiple occurrences of the same formula in a presentation are mapped onto the same image, so that no redundant images are generated. On the other hand, a caching mechanism ensures that the same equation is not rendered again in a subsequent XSLidy/FormulaTeX processing step. These mechanisms are implemented in the extension's XSLT and Perl code.

By tightly controlling the image rendering, scaling, and positioning, FormulaTeX's

results are of consistent high quality, independent from scaling issues, or the particular output environment or device.

The drawback of the  $\text{FormuL}\text{\AA}\text{T}\text{E}\text{X}$  approach is that special characters commonly used in  $\text{\AA}\text{T}\text{E}\text{X}$  equations have to be replaced by their respective entity to comply with XML syntax requirements. In particular, the characters ‘<’ and ‘&’ need to be escaped.

Once browser support for *Scalable Vector Graphics (SVG)* is stable enough, the rendering of  $\text{\AA}\text{T}\text{E}\text{X}$  equations in  $\text{FormuL}\text{\AA}\text{T}\text{E}\text{X}$  can be easily switched from bitmapped GIF/PNG images to outline-based SVG graphics. Using SVG images for  $\text{\AA}\text{T}\text{E}\text{X}$  equations would guarantee optimal rendering results for an even broader range of viewing devices and font sizes.

## 8 Future Work

XSLidy is a useful and powerful tool for managing and publishing complex presentations, but there currently is no interface for creating XSLidy files. This means that users have to manually create XSLidy XML, which is a big obstacle for the vast majority of potential users. Working towards an interface for XSLidy and providing import capabilities (for example using the XML exports of dedicated presentation software) are the two possibilities we are investigating as a way to make XSLidy easier usable for presentation authors.

## 9 Conclusions

XSLidy is a package which proves to be useful for an otherwise unsupported part of the tool landscape: the ability to create and maintain complex content for Web-based presentations. It has been in use for a year now and proves to be a very useful tool, supporting a much faster turnaround time on presentation material updates than other presentation tools. Since the package is built around the assumption that the presentations should be as Web-friendly as possible, the resulting HTML code provides high degrees of usability and accessibility.

XSLidy not only produces Web-friendly presentations, it also is built on standard technologies, most importantly XML and XSLT. One important advantage of this standards-based approach is its portability across platforms. XSLidy is actively being used on Windows, Linux, and MacOS.

## References

- [ABC<sup>+</sup>03] Ron Ausbrooks, Stephen Buswell, David Carlisle, Stéphane Dalmas, Stan Devitt, Angel Diaz, Max Froumentin, Roger Hunter, Patrick Ion, Michael Kohlhase, Robert Miner, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. *Mathematical Markup Language (MathML) Version 2.0 (2nd Edition)*. World Wide Web Consortium, Recommendation REC-MathML2-20031021, October 2003.
- [BBJJ04] Richard G. Baraniuk, C. Sidney Burrus, Don H. Johnson, and Douglas L. Jones. Sharing Knowledge and Building Communities in Signal Processing. *IEEE Signal Processing Magazine*, 21(5):10–16, 2004.

- [BC03] Robert A. Bartsch and Kristi M. Cobern. Effectiveness of PowerPoint Presentations in Lectures. *Computers & Education*, 41(1):77–86, June 2003.
- [BcHL07] Bert Bos, Tantek Çelik, Ian Hickson, and Håkon Wium Lie. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. World Wide Web Consortium, Candidate Recommendation CR-CSS21-20070719, July 2007.
- [BPSM<sup>+</sup>06] Tim Bray, Jean Paoli, C. Michael Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible Markup Language (XML) 1.0 (Fourth Edition). World Wide Web Consortium, Recommendation REC-xml-20060816, August 2006.
- [Int06] International Organization for Standardization. Information Technology — Document Schema Definition Languages (DSDL) — Part 3: Rule-based Validation — Schematron. ISO/IEC 19757-3, April 2006.
- [Kay07] Michael Kay. XSL Transformations (XSLT) Version 2.0. World Wide Web Consortium, Recommendation REC-xslt20-20070123, January 2007.
- [Lam94] Leslie Lamport. *TEX: A Document Preparation System*. Addison Wesley, Reading, Massachusetts, 2nd edition, August 1994.
- [MOV06] Jonathan Marsh, David Orchard, and Daniel Veillard. XML Inclusions (XInclude) Version 1.0 (Second Edition). World Wide Web Consortium, Recommendation REC-xinclude-20061115, November 2006.
- [Pem02] Steven Pemberton. XHTML 1.0: The Extensible HyperText Markup Language (Second Edition). World Wide Web Consortium, Recommendation REC-xhtml1-20020801, August 2002.
- [Rag06] Dave Raggett. Slidy — A Web Based Alternative to Microsoft PowerPoint. In *Proceedings of XTech 2006*, Amsterdam, Netherlands, May 2006.
- [RLJ99] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 Specification. World Wide Web Consortium, Recommendation REC-html401-19991224, December 1999.
- [TBMM04] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures Second Edition. World Wide Web Consortium, Recommendation REC-xmlschema-1-20041028, October 2004.
- [VHD06] Alexey Voinov, Raleigh R. Hood, and John D. Daues. Building a Community Modeling and Information Sharing Culture. In *Proceedings of the 3rd Biennial meeting of the International Environmental Modelling and Software Society*, Burlington, Vermont, July 2006.
- [Wil07] Erik Wilde. XInclude Processing in XSLT. *xml.com*, March 2007.