

THE

SEYBOLD REPORT

ANALYZING PUBLISHING TECHNOLOGIES

Tech Watch

What Is XSL-FO? When Should I Use It?

The answer to the first question is easy. XSL-FO stands for eXtensible Stylesheet Language formatting objects. It's a language for completely describing a styled document. The second question is not so easy, but we can oversimplify by saying that layout- and design-intensive publications are probably not good candidates for the XSL-FO approach. Content-driven documents fit the model much better, especially if they must be personalized, produced in high volume or generated in response to Web queries. If you want more depth than that, turn to this tutorial by one of the authors of the W3C specification.

A SEYBOLD REPRINT

This material appeared in *The Seybold Report* (ISSN 1533-9211), Volume 2, Number 17, December 9, 2002. Seybold Publications has prepared this reprint at the request of Adobe Systems.

©2002 by Seybold Publications, P.O. Box 644, 428 E. Baltimore Ave., Media, Pennsylvania 19063, ph. (610) 565-2480.

All rights reserved. Reproduction in whole or in part without written permission is strictly prohibited. ISSN: 1533-9211. Subscriptions are available in the U.S./Canada for \$595 per year (24 issues) and Internationally for \$640 per year (24 issues).

SEYBOLDREPORTS.com

What Is XSL-FO and When Should I Use It?

BY STEPHEN DEACH

A spate of recent product announcements regarding XSL-FO raised a number of questions in our minds: What is it? Which documents are a good fit for XSL-FO? Which aren't? What skills do publishers need to use XSL-FO? An author of the XSL-FO spec, Stephen Deach of Adobe Systems, offers these answers.

XSL stands for eXtensible Stylesheet Language. Unfortunately, the term XSL is occasionally used in both a generic and a specific sense, which leads to great confusion. Generically, XSL is actually a family of three Recommendations produced by the W3C's XSL Working Group: XSL Transformations (XSLT), XML Path Language (XPath), and eXtensible Stylesheet Language (the specific use of XSL). To ease the confusion over the specific and generic uses of XSL, most people refer to the last specification (which actually specifies the formatting objects) as XSL-FO.

There is also significant confusion over the differences between XSL-FO and CSS. CSS (Cascading Style Sheets) is an external stylesheet language. It is used to apply styling to an XML or HTML document by selecting elements in the document and attaching styling properties to each selected element. In contrast, XSL-FO is a language for completely describing a styled document, including its content organization, styling, layouts and layout-selection rules—everything needed to format and paginate it. To use it, one applies an XSLT stylesheet (or some other mechanism) to the original XML or XHTML document, transforming into an XSL-FO document, which is then fed to a formatter.

XSL-FO's history and status

The XSL Working Group (XSL-WG) was initially chartered by the W3C (World Wide Web Consortium) in January 1998. The first official WG meeting was held in February 1998.

In early 1999, it was decided to split XSLT and XPath out into separate specifications, since they are used in other environments within W3C and could be made available sooner than the formatting objects. These both became W3C recommendations on November 16, 1999.

The formatting portion became a W3C recommendation on October 15, 2001.

Why was XSL developed? At the time the XSL-WG was formed, there were three significant issues with the existing Web standards.

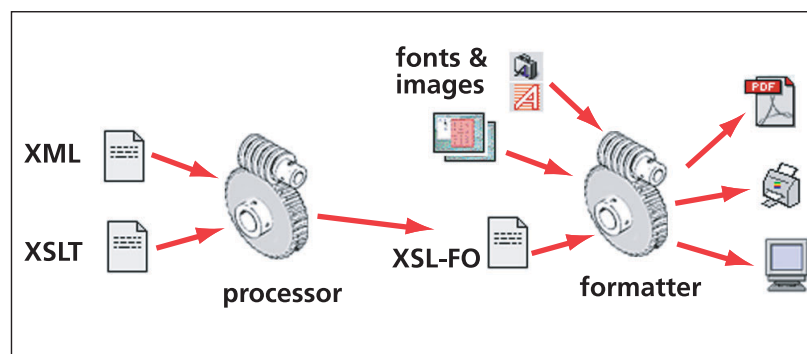
First, there was no language for describing the pagination of complex documents on the Web. (Even today, when you print a document from the common-brand browsers, the line of text at the page boundary is often sliced in half. If this happens in a picture, half the picture appears on the first page and a blank box appears on the next page.) CSS-2.0 added some basic support for pagination, but to this day it has not been fully implemented in the common browsers. Moreover, CSS will work only with documents whose content is organized the same way as the presentation—no skipped elements, no elements presented out of sequence, etc.

Second, there was no way to deal with long documents and complex layouts. No Web standard could support things like tables of contents, top and bottom floats, footnotes, endnotes, indexes or multiple articles on a page—layouts that are common in newspaper, magazine or retail-catalog pages.

Third, the level of typography in CSS was not sufficient for print documents. CSS was limited to what was needed for browsers and easy for the browser manufacturers to implement.

Goals of the Working Group

To respond to the issues above, the XSL-WG established a number of goals for the design of XSLT and



Production pipeline. XSL-FO is an intermediate form between media-neutral XML and media-dependent output. You feed your XML structured content and an XSLT stylesheet to an XSLT processor. The result is XSL-FO. Then you feed the XSL-FO, along with font metrics and any external graphics, into an XSL-FO formatter. The result is a paginated document (in PDF or printer-native code) that can be displayed or printed.

Glossary

A unique jargon grows up around many standards, and XSL is no exception. Here are some plain-language explanations for a few key XSL terms. These explanations lack formal rigor, but we hope they convey a sense of the roles of the various parts of the XSL standard.

XSLT (XSL Transforms): The word “Transforms” accurately captures what this part of the standard is about: It describes how XML documents are to be filtered and converted into other XML documents, including XSL-FO files. Unfortunately, an XSLT that specifies this type of conversion is called a “stylesheet,” which is a confusing extension of this traditional typesetting term. An XSLT stylesheet is closer in concept to a sophisticated search-and-replace routine than it is to the stylesheets of the past.

Xpath (XSL Path Language): This is used within XSLT to specify the parts of an XML document to which transformations are to be applied. XPath models an XML document as a tree of nodes with parent-child and next-previous relationships. These nodes are of different types: element nodes, attribute nodes, text nodes, etc. If XSLT

is a sophisticated search-and-replace facility, then XPath is the way you select which nodes or types will be searched.

XSL-FO (XSL Formatting Objects): The XSL-FO file contains the medium- and appearance-specific “formatting objects” that make up the page (or, for audio output, the speech). For the print medium, formatting objects can include characters, blocks of text, images, tables, borders, master pages and so on.

XSL-FO is not a page-description language. It can specify various layout rules (e.g., where page breaks can occur) and requirements (e.g., footnotes go at the bottom of the page), but it does not determine the actual placement of each element. That is determined by the XSL-FO pagination engine, called a **formatter**.

The output from a formatter need not actually drive a printer. Rather, the output might be a PostScript or PDF file, which would need additional rendering software. Indeed, an earlier article by Bernd Zipper (see *Vol. 2, Nos. 10 and 11*) described several XSL-FO formatters that are commonly used to generate PDF on-the-fly.

George Alexander

XSL-FO. I’ve summarized the five most important ones below.

1. XML syntax. XSLT and XSL-FO should be expressed in a pure XML syntax. This has two significant advantages: XSLT and XSL-FO work with existing XML parsers, and they can be validated and manipulated using existing XML tools.

2. Declarative. XSLT and XSL-FO should be declarative, a set of elements and attributes fully describing what the desired result looks like. (In contrast, a procedural language such as PostScript describes the algorithms that will yield the result.) Even though programming expressions are allowed in some attributes, there should be no requirement to use them and no requirement to use internal scripting.

3. Build on CSS. XSL should extend and augment CSS-2 to cover a different problem-space. At the same time, it should share as many styling properties and as much of the formatting/layout model from CSS as possible.

The extended problem-space includes support for paginated documents with greater complexity than can be supported by decorating (attaching styling attributes to) the original XML-content tree. For content-driven documents (such as textbooks, manuals, contracts or industrial catalogs), XSL should add support for features like the aforementioned footnotes, endnotes, floats, indexes, tables of contents, odd and even page masters, insert pages, running headers and footers, etc. It should also add support for layout-driven documents such as newspapers, magazines, catalogs and brochures. Of course, it should continue to support the scrolling-galley docu-

ments (browser documents) that could be supported using CSS. (For a more in-depth discussion of layout-driven vs. content-driven documents, see my presentation from Seybold San Francisco ‘98 at www.seyboldreports.com/TSR/free/0217/sdeach.pdf.)

4. Cross-media. XSL should cover the basic presentation requirements for most print uses; for a wide range of display devices, including reflow or repagination for palmtop devices; and for the accessibility requirements that are now mandated by many governments.

To do this, it is necessary to provide a transformation mechanism to reorganize and filter the content to present a better representation for each desired medium and access requirement. This also makes it necessary to separate out the layout designs and the pagination sequencing controls so that the same stylized content can be placed in different layouts.

The transformation mechanism should also provide a way to modify the presentation (styling and layout) to support accessibility requirements such as larger text or alternate fonts, black-on-white and white-on-black (or alternate color schemes), alternate navigation and aural presentation.

5. Typographic quality, multilingual market. XSL-FO should match or exceed the typographic and layout features of existing page formatters and should be compatible with their underlying formatting models.

XSL-FO’s multilingual support should include the use of Unicode and the Unicode bidirectional rules. It should support a number of the special composition features used in Asian and Middle Eastern languages, or at least to make it easy to add them in the future. XSL-FO should also provides support for the multilin-

gual features of modern font technologies, such as OpenType fonts.

Intended-use guidelines. In addition to the goals for the project, the Working Group also established some guidelines or expectations for how XSL-FO would be used. Most important, XSL-FO is explicitly *not* designed to be hand-authored. Rather, it is supposed to be machine-generated through XSLT, scripting, and composition or design applications. This is not a radical concept; today's dynamically published Web sites routinely store their content in neutral XML and then machine-generate the appropriate HTML for the target browser.

Though not a primary goal, it was expected that XSL-FO could also serve as an interchange language for prestyled content.

How close did XSL come to its initial goals? Since it was necessary to make the final-feature decisions for XSL-FO far enough in advance to allow for production of a CR (Candidate Recommendation) document, as well

as to allow for about a year of interoperability testing between several independent implementations, the feature set was frozen in February 1999. In order to meet a fall-1999 CR date, it was decided to deliver only content-driven formatting in the initial release, deferring layout-driven formatting, full internationalization and forms support to a later release.

The initial spec for XSL-FO had basic support for multilingual documents, including support for bidirectional and vertical writing modes. There had been some overlapping work in other Working Groups dealing with full internationalization and form fields, but these joint efforts were not far enough along when XSL-FO became a candidate recommendation. Therefore, it was necessary to defer support for full internationalization and forms.

Except for these deferrals, we met our original goals.

Current activity

Activity is ongoing, even as the XSL Working Group goes through the process of rechartering. (Working

Degrees of XSL-FO support

XS�-FO is picking up momentum, with various developers adding support for it. Here are some of the implementations:

- **FOP**, Apache (xml.apache.org/fop). Open source.
- **PassiveTeX**, TEI (www.tei-c.org.uk/Software/passivetex). Open source.
- **XSL Formatter**, AntennaHouse (www.antennahouse.com). Commercial product.
- **XEP**, RenderX (www.renderx.com). Commercial product.
- **E3**, Arbortext (www.arbortext.com). Support incorporated into existing commercial product.
- **XSL-FO Renderer**, Advent (www.3b2.com). Forthcoming "free or nearly free" product (Spring 2003).
- **Adobe Document Server 5.0**, Adobe Systems (www.adobe.com). Commercial product.
- **Infoprint XML Extender for z/OS**, IBM (www.printers.ibm.com/R5PSC.NSF/Web/xmlextenderhome). Commercial product.

All XSL-FO support is not alike. First, there are the degrees of conformance that the standard itself sets out. The Basic level of conformance supports the formatting objects and properties "needed to support a minimum level of pagination or aural rendering." The Extended level includes capabilities for "applications whose goal is to provide sophisticated pagination." The Complete level adds "shorthands." An example of a shorthand

is `border="thin,red"`, which sets all four border-width properties (left, right, top, bottom) to thin, all four border-color properties to red, and (implicitly) all four border-style properties to solid.

Second, the available implementations of XSL-FO do not fall neatly into these three levels. Many implementations do not even cover all the Basic features. For example, the print-oriented renderers generally omit the aural (speech synthesis) aspects. Vendors may also omit some of the more "exotic" aspects of the standard for which they don't anticipate an actual need—or don't already support in their existing formatting engines. Most implementations do not support any writing-mode except left-to-right, for example, and Arbortext will not have mixed-language hyphenation in its initial offering. The Web sites for the two open-source implementations (Apache's FOP and TEI's PassiveTeX) and for the first two commercial implementations (XSL Formatter from AntennaHouse and XEP from RenderX) provide exhaustive detail about which parts of the basic and extended feature set these implementations support.

There are also features missing from the initial XSL-FO specification that some publishers will find essential. For example, the spec has only limited support for "floats" (e.g., illustrations that are automatically placed at the top or bottom of the page where they are referenced), so the Adobe and Arbortext implementations will use the namespace mechanism of XML to introduce private tags and private attributes to go beyond what the specification requires. Advent notes that XSL-FO's support for table-of-contents generation is very limited and its facilities for index generation are not sufficient for even a moderately complex back-of-the-book index. But Advent will probably not add these capabilities to its forthcoming XSL-FO renderer. Those who need them will be pointed toward Advent's commercial product, the 3B2 publishing system.

George Alexander

Groups in the W3C are always chartered for a finite agenda and a finite period of time.) On the XSL-FO side, the members are in the process of following up on errors and omissions in the XSL 1.0 document. They have published an errata document and responses to the questions posted to the comment lists. They are proposing an XSL 1.1 document to correct errata and make some additions to support dynamic documents. They are also gathering requirements for XSL 2.0.

On the XSLT and XPath side, the WG is in the process of developing version 2.0. It has recently published updated working drafts on each of these.

Future directions. The first priority is to correct problems discovered in the current document. Then, it is certainly expected that the XSL WG will address the capabilities that were deferred from XSL 1.0 due to time limitations. These include layout-driven documents, full internationalization and named stylesets.

Why and when should I use XSL-FO?

I frequently get asked, “Given CSS, why do I need XSL-FO?” Another common question is, “Given the wonderful WYSIWYG composition tools we already have, many of which can process XML, why bother with XSL-FO?” Some questioners also note that CSS and XSL stylesheets are arguably harder to write (and certainly less designer-friendly) than the nice GUIs of composition programs such as FrameMaker and InDesign.

If I ever publish a book on composition technology, I’m going to subtitle it, “There is never just one right way.” The requirements of the publishing industry are broad and diverse. Some documents require the creative professional to interactively modify the layout, the content and the typography to get an acceptable result; other documents are best generated and formatted on demand using rule-based systems. XSL-FO is designed to fill a hole in the format-on-demand sector of this industry; thus it is an ideal fit for batch-paginated products such as Adobe Document Server.

The requirements of the format-on-demand sector are quite different from those addressed by interactive, WYSIWYG composition tools such as FrameMaker and InDesign. Today’s WYSIWYG composition tools are very good at creating and perfecting individual documents, which is the primary thrust of much of the traditional publishing industry. In these WYSIWYG products, the content and styling and layout are tightly bound within the document, so reflowing the document to an alternate medium (display vs. paper, displays with differing sizes or form-factors) is difficult.

Another area where WYSIWYG tools are not so good is producing thousands of documents whose content is pulled from a database and poured into templates. Examples include insurance policies, contracts, manuals, mail-order catalogs with regional or demographic variations, and textbooks and teaching aids with state-by-state or school-district-specific varia-

tions. In circumstances like these, the important issue is the ease with which whole collections of documents can be given a common look and feel.

Where interactive is best. There are two situations that are best handled by interactive desktop-composition software: when a designer needs creative control of the layout, and when a writer or illustrator needs to work on the content while simultaneously viewing or modifying its layout. In magazine production, for example, page design is often complex and multilayered, and the layout for each page is uniquely tailored to the content. (Often, both the content and the layout will be adjusted repetitively to get the desired result.) In brochure production, the graphics and the layout may be produced at the same time and changes in one influence the other. In newspaper production, content is often edited to fit an exact space. In these situations, interactive composition tools like Adobe InDesign and Adobe InCopy are proper choices.

The workflow for creating technical documentation (and other types of long documents) is rather different; because of the volume of the content, the styling and layout are often applied via rules. The styling provides the reader with visual cues of the content’s organization and the specific meaning of some text (such as commands or language scripts), so it helps the author’s creativity and improves the document’s quality if the author can see the content being styled as it is written and edited. And it also helps to see the pages laid out as a reader would see them, as this allows minor adjustments to be made to improve the overall result. Therefore, using interactive composition software for such publications still makes sense, although the ideal tools would still start with a rules-based approach to page layout; this is the approach taken in Adobe FrameMaker.

Where XSL-FO is best. There are situations, however, where interactive composition is impossible. Examples include generating content-driven documents in response to individual customer requests (perhaps coming to your Web server) or variations of a document to match different demographic sectors. In such situations, XSL-FO running in a product like Adobe Document Server is the proper choice.

Although the range of applications is expected to grow over time, at this point a server-based XSL-FO solution is useful for producing documents such as:

- **Financial-planning guides.** These are custom reports that recommend stocks, bonds and mutual funds, each with an associated prospectus. The versioning is specific to each customer, financial planner and stockbroker.
- **Owner and maintenance manuals.** Increasingly, these are generated based on a specific user request

and customized to include only the information that is relevant to the customer's model or serial number.

- **Legal agreements and contracts.** Initially, paralegals will produce a *pro forma* contract with boilerplate from a legal-agreement database. During the negotiations, they repeatedly produce documents that represent the current state of the contract with all information up to date and in one place. Only at the very end do they produce an “as signed” archival version with the terms that were accepted by the principals.

Another important goal is to print uniquely created content that takes graphics and text from the same neutral XML content sources that are driving dynamic publishing on the Web. XSL-FO, in conjunction with its companion XML specification SVG, gives a clear way to unite Web-oriented dynamic publishing tools and print.

XSL-FO vs. an existing batch solution. If you have a batch system that you are happy with, today is not the day you want to move it to XSL-FO. In fact, since you have a tested solution and your people are trained on how to use it and how to work on it, you may not ever want to migrate it to XSL-FO.

On the other hand, if you do not currently have a batch solution, if you are looking at a new problem, or if you are expanding into new markets; then the decision becomes more difficult. Many of the batch vendors will be adding support for XSL-FO as an adjunct to their native formatting language or their built-in XML solution. (A sidebar on page 5 describes some of today's products.) The decision comes down to the fit of a particular language and workflow to the problem at hand. This must be evaluated on a case-by-case basis.

I can give only some general suggestions. The process can be divided roughly into two categories: preparing your content for composition, and matching your formatting requirements to composition functionality.

Content preparation usually entails a conversion from some original data format into something you can feed to the XSL-FO formatter. You will need to do one of three things: Write a program in a traditional programming language, write a script in one of the many scripting languages, or use XSLT. Your choice must be based on the skills you have available or can acquire.

Matching your composition problem to the styling mechanism requires first that you find a formatting engine that covers the functional requirements you have. Then you must evaluate the language and styling mechanism that you use to direct that engine. (Do you really need every feature of the composition engine or

of the styling language? Probably not.) If your suppliers are providing the styled input, you must also evaluate the language and styling mechanism based on their abilities.

As I said before, different solutions work best for different problems. Some people will be more comfortable in an XML-tools and XSLT environment, others in a programming-language or script-based conversion environment. Some will be more comfortable moving through an intermediate language such as XSL-FO, while others will be more comfortable driving their batch system in its native language.

How do I get started?

XSL-FO is now in the “early-adopter” phase. This means the user must have higher skills than will be required once it becomes well established. Fortunately, the people who are already working with server-based document-production systems probably have most of the necessary core expertise.

For those that want to move into the use of XSL-FO, I'm going to point you to the W3C's XSL Web page (see sidebar on this page). I suggest you look at the tutorials on XSLT and XSL-FO that are cited there and choose the one that seems a best-fit for your level of understanding and for your usage environment.

Further Information

Start with the W3C's XSL information page at www.w3.org/Style/XSL for general information. This page contains news on XSL (XSLT, XPath and XSL-FO), as well as pointers to books, tutorials, training companies, products, mailing lists and discussion boards where you can get further info or assistance.

Links to the XSL specifications. For an introduction, you should to read one of the tutorials or textbooks describing XSLT and XSL-FO. The specifications listed below are highly technical; they are intended as reference documents for the software developer.

XSL Transformations (XSLT), Version 1.0. XSLT is a transformation language. It is designed for writing filters that convert one XML document to another. XSL-FO is one possible output that can be generated by applying an XSLT stylesheet to an XML document. The specification can be found at www.w3.org/TR/1999/REC-xslt-19991116.

XML Path Language (XPath), Version 1.0. XPath is used within XSLT to control the traversal of the input tree and to select elements for processing. The specification can be found at www.w3.org/TR/1999/REC-xpath-19991116.

Extensible Stylesheet Language (XSL), Version 1.0 (aka XSL-FO). XSL-FO (XSL Formatting Objects) is an XML-syntax language that describes a “vocabulary for specifying formatting semantics.” It consists of a set of XML elements (formatting objects) and associated XML attributes (styling properties) that define the input to a formatting engine. The specification can be found at www.w3.org/TR/2001/REC-xsl-20011015.

Information on CSS. See the similar W3C information page on CSS at www.w3.org/Style/CSS.

TSR

The tools are coming, the tools are coming . . . It is not intended that XSL-FO be manually created in a basic text editor (the way most early HTML and CSS was created). Instead, it is expected that a wide variety of authoring tools will become available over the next few years. Some of these will evolve from basic database-driven tools that run scripts or XSLT to produce the XSL-FO. Others will be plug-ins for today's interactive composition tools that allow you to capture the stylesheets and flow the content. Eventually, there will be interactive tools designed specifically for creating XSLT stylesheets and XSL-FO stylesheet components.

Hand-authoring XSLT and XSL-FO. As noted above, XSLT and XSL-FO were designed for machine generation, not manual authoring. Nevertheless, authoring is not impossible to do. Since it is an XML-syntax, XSL-FO can be created through nearly any of the XML editing tools. Once you get through the initial learning curve, it is not significantly more difficult to author XSLT than it is to author CSS-2 selector rules of equal sophistication.

The mapping from XML+CSS to XSL-FO is pretty straightforward. For nearly all values of the CSS "display" property, the display property's value becomes the XSL-FO element tag. The XML element tag becomes the value of the XSL-FO role property, and the remaining properties are changed from CSS's `style="name:value"` syntax to XML's `name="value"` syntax.

Converting HTML+CSS adds only one further task: One must provide explicit styling properties for each HTML element tag that has implicit properties.

In content-driven print-production environments that have hierarchical styling models (such as those commonly used with SGML), users will have no trouble adapting to the use of XSL-FO. It is only necessary to learn XSL-FO's element names, attribute names and attribute-value names.

Those people who are currently using traditional markup languages (originating in the late 1960s and remaining the primary formatting language model

through the mid-1980s) will need to learn to adapt to a hierarchical model.

Patience, patience. XSL-FO has been a recommendation for only about a year. We are just starting to see the first tools come on the market that simplify its authoring. This is about the same rate of progress as with later versions of HTML, and is faster than we saw with CSS. It took about four years for CSS-1 to be fully adopted. As Web standards become more sophisticated, the adoption rates become slower. For example, the adoption of the majority of CSS-2 is progressing at a somewhat slower rate than CSS-1. Therefore, I expect the significant adoption of XSL-FO to develop over a three-to-five-year time frame.

Conclusion

XSL is a set of three W3C Recommendations (commonly referred to as XSLT, XPath and XSL-FO) that define a stylesheet language for producing format-on-demand documents. It is not a replacement for your WYSIWYG authoring tools, but it is useful for some problems, such as very large documents or those derived from database content, that are not well served by the current tools. Or at least it will be. At this time, one year after XSL-FO was accepted by the W3C, there are not a large number of tools to make authoring of XSL-FO easy. They are beginning to appear, though, and we have recently seen a flurry of announcements. Today, XSL is most useful if you need to produce customer-tailored, paginated documents on a server. I expect this to broaden over the next few years.

TSR

About the Author

Stephen Deach is a Senior Computer Scientist at Adobe Systems. He has represented Adobe on the W3C-XSL Working Group and was editor of the formatting objects portion of the XSL specifications during most of the development of XSL-FO. He has been a speaker at a number of Seybold conferences and has written an article on font-hinting technology for *The Seybold Report on Desktop Publishing* (see Vol. 6, No. 7). At Adobe, he has worked on Illustrator, FrameMaker, Acrobat and the Adobe Document Server. Contact him at sdeach@adobe.com.