



Why Johnny can't read XML

Shedding a Little Light on XML

From what I've seen, people go through a lot of pain when learning XML, probably for several reasons. The most prohibitive? Learning XML really does involve... learning XML – in addition to XML DOM, DTDs, XSLT, and possibly many other members of the XML family. It's difficult to know where to begin, and learning them all at once can be a bit overwhelming.

Just knowing XML is a good start. At the same time, simply knowing that XML is a robust, text-based way of representing information isn't enough. You need to know how to do something with it. This is where the XML DOM may come into play. It's quite powerful and not that difficult to learn, though once the concept of nodes is thrown into play most people have painful flashbacks of their Data Structures class. However, with a little motivation, it's possible to get past that and start looking at nodes with a different eye. In college I hated working on algorithms involving node sorting, but when I poked around in the XML DOM for a few days, I found it a much easier, and more powerful, approach than what was covered in my college years.

The unwanted second cousin of XML, DTDs, come into play next. In their own right DTDs are a necessity for anyone using XML for a B2B, B2C, or any other data-exchange project. However, because they come from the days of SGML, they also inherit their complexity. Okay, so in the grand scheme of things (no pun intended) they're really not that complex. They just seem that way because they're in a different syntax. I happen to like working with DTDs, and find that they allow concise representation of the structure of an XML document. Although I wasn't a fan at first glance, I knew I had to learn them. And that's precisely what I did.

It's likely that most folks take a mental step back when looking at DTDs, thus ruining the momentum of the learning curve. More recently, XML Schemas have been approved as a W3C recommendation and Relax-NG has been started as an OASIS Technical Committee project. I

think that both will change the way people view this particular aspect of XML.

One way to view these different modeling approaches is that:

- DTDs work with both the syntax and information set of an XML document.
- XML Schema works entirely with the information set of an XML document.
- Relax-NG works entirely with the syntax of an XML document.

The choice between the two new approaches depends on how you need to work with your information. Aside from offering greater power and flexibility over DTDs, both are written in XML – a significant achievement that should lighten the pain and suffering for newbies and experienced geeks alike.

Last but not least is XSLT. This third area of XML is undoubtedly my favorite, but for many it's the one that needs the most hand-holding. Oddly enough, I took an immediate liking to it. I don't know why. Really. (It could be a genetic flaw...or it could be an interest in the brain teaser-like nature of XSLT. I may never know.) Unfortunately, I haven't found the same reaction in enough of my colleagues. This might be motivational, though, as once we spent some time together working on XSLT they really started to enjoy it.

One part of the problem is that few are aware of what XSLT is capable of. Its power isn't self-evident; you have to dig into it a bit. Those who have dug are very excited by it. It's hard not to be. Another part of the problem is that few see the importance of XSLT until it's too late for the learning curve to be factored in. If you are one of these people, put down this article right now and pick up a copy of Michael Kay's XSLT pro-

grammers' reference, Crane Softwrights' electronically published tutorial (www.CraneSoftwrights.com/links/trn-xps.htm), some of the online classes at www.vbxml.com/trainit/, or Infoteria's Style-sheet Central (www.infoteria.com/xsl).

XSLT is a new animal altogether, yet it's mostly a new approach to an old problem: "What can I do with my XML?" A strong knowledge of XML simply as a data source is similar to a strong knowledge of how to look at a table in Access or SQL Server Enterprise manager. Unless it can be transformed into another form, be it HTML or another format of XML, you're limited in what you can do with your XML. In fact, XSLT is really a new templating language – one that isn't procedural, but rather a functional language without an assignment statement. The lack of a statement has raised as much controversy and skepticism as did the famous article by Edsger Dijkstra, "GoTo Statement Considered Harmful."

Additionally, the idea of template-oriented programming is one that most people like to avoid. It took me some time to catch up to this concept rather than solving every XSLT-related challenge using the Pull method.

This month I'll touch on some of the questions related to the roadblocks described above. As you read questions from other XML developers, I hope it will become apparent that we've all gone through the same thought processes and asked ourselves the very same types of questions.

FAQ

Q: *When and why should templates be used in XSLT?*

A: First, let me introduce two new terms: the *Pull* method and the *Push* method. These terms

AUTHOR BIO

David Silverlight, chief XML evangelist for Infoteria, has been working in the trenches for a number of years as a software architect and consultant, specializing in database-driven Web applications. He also maintains www.xmlpitstop.com, a resource for XML examples and everything else XML.

simply describe different techniques of XSLT stylesheet design. Sometimes just getting a new view on an existing problem is all it takes to make them “click.” As mentioned earlier, template-oriented XSLT tends to hinder the learning process a bit. For me it took a brilliant class by Ken Holman for this concept to sink in. Hopefully they’ll help to appease the fears of those who are new to XSLT development. They’re defined as follows:

- **Push method:** Essentially, an approach in which the processor searches for the template that describes the result of processing an element and attribute from a collection of template rules that may match the type of information. A visual mechanism might be to think of it as: “I have defined my templates and I would like to Push data toward it.” This approach decomposes the XSLT task into a number of smaller templates.
- **Pull method:** An approach involving directly supplying the template that describes the result of processing an element and attribute. Using Pull exclusively allows you to (1) define only the required root template rule and (2) use XSLT programming constructs for all remaining transformations. Think of this method as: “I have an XML document and I wish to Pull data from it.”

Which One Should I Use?

Just as there are many ways to write the same computer software, there’s more than one methodology to technically produce a fully functional stylesheet. A lot of unprofessional code is out there, in all languages. There’s code that works but is difficult to maintain, code that breaks at intermittent points, and code that’s never seen an error-handling routine. All of the above-mentioned examples may appear to work fine on the surface. As with the Push versus Pull comparison, these negative examples are worth mentioning since the conditions usually arise from programming style...and sometimes just from mere laziness. Essentially, your choice of method boils down to programming style.

Using the Pull method exclusively can lead to a single stylesheet – possibly hundreds or thousands of lines long – that will function fine but may not allow code reuse and/or maintainability. As Steven Wright would say: “Hard work pays off in the future. Laziness pays off now.” Beginner XSLT enthusiasts may use this approach to develop their stylesheets since it allows the use of common programming constructs like if/endif, for-next, and select right off the bat before you’ve gained an understanding of templates.

The Push or template-oriented approach to developing stylesheets will likely require a bit more work up front, but it’s the true nature of developing in XSLT. Instead of creating a single template rule and doing all of the work inside that rule, you break it down into a number of templates. XSLT is a template-based programming language and should be treated as such. Deviations from this approach will restrict you from getting into the true feel of the language. More important, creating a series of templates allows you to break down the solution into smaller modules whose logic may be applied and reused in future stylesheets.

The bottom line is that this is a question of who or what is in charge of the result: use the Pull method if the XSLT stylesheet (or stylesheet author) is in charge, and use the Push method if the XML data (or data author) is in charge.

FAQ

Q: *If there’s no assignment operator, what can I use in its place?*

A: XSLT programming is more than just a new templating language. In fact, it’s a paradigm shift from the way many of us approach

problem solving. As mentioned earlier, the assignment statement is noticeably absent in XSLT. This results in the question: How do I assign a value to a variable? In procedural programming language it might be done as follows:

```
int InitialRate;
InitialRate = 25;
```

But in XSLT it could be accomplished through the construct of

```
<xsl:variable
name="InitialRate"
select='25'/>
```

On the surface this appears to accomplish the task and works quite satisfactorily for setting an initial value. It behaves much differently than might be expected if it’s compared to a procedural programming language. In XSLT this value can be initialized, but it can’t be changed via a subsequent variable statement the way most view an assignment statement. Often it’s necessary to perform a conditional assignment. For example:

```
var InitialRate = 0;
if (JobTitle="programmer")
{
    InitialRate=125;
}
else {
    InitialRate=25;
}
```

If the XSLT version seems more verbose, it’s because it *is* more verbose as a by-product of the use of markup for the expression language. Once again, it’s just the nature of the beast. The XSLT construct for a conditional select could be done as follows:

```
<xsl:variable name="InitialRate">
<xsl:choose>
<xsl:when test="$JobTitle =
'programmer'">125</xsl:when>
<xsl:otherwise>25 </xsl:otherwise>
</xsl:choose>
</xsl:variable>
```

When iterating through data with a programming language:

```
int chapNo;
...
chapNo = chapNo + 1;
```

AD

When templating, the XSLT processor does the dirty work and the writer just has to declare the work needed of the processor to accomplish the task.

```
<xsl:number count=
  "chapter" from="section"/>
```

Black Belt

Q: *How do I compare complete nodesets?*

A: Most people get flashbacks of “Data Structures 101” when they first realize that nodesets are a key aspect of XML and XSLT. Nodesets shouldn’t be fretted over, and there are a few tried and true algorithms for finding the intersection and difference of nodesets. Following are some of the most common algorithms.

- **Kaysian Intersection:** Returns the intersection of two nodesets:

```
$nodeset1[count(.|$nodeset2)=
  count($nodeset2)]
```

- **Becker Difference:** Returns the difference between two nodesets (actually a variation of the Kaysian Intersection):

```
$nodeset1[count(.|$nodeset2)!=
```

```
count($nodeset2)]
```

- **Holman’s Difference:** Returns the symmetric difference between two sets:

```
($nodeset1 [count(.|$nodeset2)!=
  count($nodeset2)] |
  $nodeset2 [count(.|$nodeset1)!=
  count($nodeset1)] )
```

Note: The differences between “difference” and “symmetric difference” are as follows:

- **Symmetric difference:** Contains elements that are in either Nodeset1 or Nodeset2, but not both
- **Difference:** An element is in Nodeset1 but not in Nodeset2

The downloadable source code for this article contains examples of each of these algorithms in use by demonstrating examples of customers who shopped in Store A and Store B.

Elements of Design


Seeing Is Believing: An XSLT Stylesheet Repository of Enormous Magnitude

As a developer I have found that one of the quickest ways to learn a new technology is by examining working examples to see what happens “behind the scenes.” In the world of XSLT this can be even more

essential. A true gold mine of examples of XSLT stylesheets can be found at Infoteria’s XSL Stylesheet Central (www.infoteria.com/xsl). Here I’ve developed a huge repository of examples (growing every day) of how to do everything from the simplest task of displaying an HTML table in alternating colors to more advanced functionality like grouping XML data together using the Muenchen Method. The best perk of this site is that all of the source can be downloaded to see *how* it’s done. In addition, all of the XSLT code is conveniently deconstructed and analyzed to explain how and why techniques are used. It’s a definite “must see” for serious XSLT developers and beginners alike.

• • •

This monthly column is an opportunity for me to respond to the questions that tug at our brain cells and beg to be answered. So send them in my direction at dsilverlight@infoteria.com. I’ll select those that fall into one or more of the following categories:

- **FAQ:** What you always wanted to know
- **Black belt:** The toughest of the tough
- **Customizing XSLT:** My personal favorite and by far the most popular
- **Going wireless:** Bringing the world of wireless to your eyes 

DSILVERLIGHT @ INFOTERIA.COM

Ad

www.ad.com