....
Search
Advanced search

IBM home | Products & services | Support & downloads | My account

IBM developerWorks : XML zone : XML zone articles

developerWorks

What kind of language is XSLT?

PDF e-mail it!

An analysis and overview

Michael Kay (Michael.Kay@softwareag.com)
Software AG
February 2001

What kind of a language is XSLT, what is it for, and why was it designed the way it is? These questions get many different answers, and beginners are often confused because the language is so different from anything they are used to. This article tries to put XSLT in context. Without trying to teach you to write XSLT style sheets, it explains where the language comes from, what it's good at, and why you should use it.

Contents:

Related content:

Transforming XML documents
Improve your XSLT coding five ways
More dW XML resources

I originally wrote this article to provide the necessary background for a technical article about Saxon, intended to provide insights into the implementation techniques used in a typical XSLT processor, and therefore to help users maximize the performance of their style sheets. But the editorial team at *developerWorks* persuaded me that this introduction would be interesting a much wider audience, and that it was worth publishing separately as a free-standing description of the XSLT language.

What is XSLT?

The XSLT language was defined by the World Wide Web Consortium (W3C), and version 1.0 of the language was published as a Recommendation on November 16, 1999 (see Resources). I have provided a comprehensive specification and user guide for the language in my book *XSLT Programmers' Reference* and I don't intend to cover the same ground in this paper. Rather, the aim is simply to give an understanding of where XSLT fits in to the grand scheme of things.

The role of XSLT

XSLT has its origins in the aspiration to separate information content from presentation on the Web. HTML, as originally defined, achieved a degree of device independence by defining presentation in terms of abstractions such as paragraphs, emphasis, and numbered lists. As the Web became more commercial, publishers wanted the same control over quality of output that they had with the printed medium. This gradually led to an increasing use of concrete presentation controls such as explicit fonts and absolute positioning of material on the page. The unfortunate but entirely predictable side effect was that it became increasingly difficult to deliver the same content to alternative devices such as digital TV sets and WAP phones (*repurposing* in the jargon of the publishing trade).

Drawing on experience with SGML in the print publishing world, XML was defined early in 1998 as a markup language to represent structured content independent of its presentation. Unlike HTML, which uses a fixed set of concepts (such as paragraphs, lists, and tables), the tags used in XML markup are entirely user defined, and the intention is that they should relate to objects in the domain of interest (such as people, places, prices, and dates). Whereas the elements in HTML are essentially typographic (albeit at a level of abstraction), the aim with XML is that the elements should describe real-world objects. For example, Listing 1 shows an XML document representing the results of a soccer tournament.

Listing 1. An XML document representing the results of a soccer tournament

```
<results group="A">
<match>
    <date>10-Jun-1998</date>
    <team score="2">Brazil</team>
    <team score="1">Scotland</team>
</match>
<match>
    <date>10-Jun-1998</date>
    <team score="2">Morocco</team>
    <team score="2">Norway</team>
</match>
<match>
    <date>16-Jun-1998</date>
    <team score="1">Scotland</team>
    <team score="1">Norway</team>
</match>
<match>
    <date>16-Jun-1998</date>
    <team score="3">Brazil</team>
    <team score="0">Morocco</team>
</match>
<match>
    <date>23-Jun-1998</date>
    <team score="1">Brazil</team>
    <team score="2">Norway</team>
</match>
<match>
    <date>23-Jun-1998</date>
    <team score="0">Scotland</team>
    <team score="3">Morocco</team>
</match>
</results>
```

If you want to display these soccer results through a Web browser, you can't expect the system to come up with a sensible layout. Some other mechanism is needed to tell the system how to display the data on a browser screen, a TV set, a WAP phone, or indeed on paper. This is where the style sheet comes in. The style sheet is a declarative set of rules that defines how information elements identified by tags in the source document should be rendered.

The W3C has defined two families of style sheet standards. The first, known as CSS (Cascading Style Sheets), is widely used with HTML, though it can also be used with XML. CSS can be used, for example, to say that when displaying an invoice, the total amount payable should be shown in 16 point Helvetica bold. However, CSS has no ability to perform computations, to rearrange or sort the data, to combine data from multiple sources, or to personalize what is displayed according to characteristics of the user or session. In the case of our soccer results, CSS (even the latest version, CSS2, which is not yet fully implemented in products) is not a powerful enough language to handle the task. For these reasons W3C embarked on the development of a richer style sheet language to be known as XSL (Extensible Stylesheet Language), taking many of the intellectual ideas from DSSSL (Document Style, Semantics, and Specification Language), as developed in the SGML community.

During the development of XSL (and this had already been foreshadowed in DSSSL), it emerged that the tasks to be performed in preparing an XML document for display could be split into two stages: transformation and formatting. Transformation is a process of converting one XML document (or its in-memory representation) into another. Formatting is the process of converting the transformed tree structure into a two-dimensional graphical representation, or perhaps a one-dimensional audio stream. XSLT was developed as a language to control the first stage, transformation. Development of the second stage, formatting, is work still in progress. But in practice, most people are currently using XSLT to transform XML documents into

HTML, and using an HTML browser as the formatting engine. This is possible because, for all intents and purposes, HTML is just one example of an XML vocabulary, and XSLT is capable of using any XML vocabulary as its target.

Separating transformation into one language and formatting into another proved to be a really good decision, because it turned out that there are lots of applications for a transformation language that have nothing to do with displaying documents to the user. As XML becomes more widely used as a data interchange syntax in electronic business, there is an increasing need for applications to convert data from one XML vocabulary to another. For example, an application might extract details of TV programs from an electronic program guide and insert them into a monthly bill for a pay-per-view customer. Equally, there are many useful data transformations in which the source and target vocabularies are the same. These include data filtering, as well as business operations such as applying a price increase. Increasingly therefore, as data starts to flow around the system in XML syntax, XSLT starts to become a ubiquitous high-level language for manipulating it.

In my book I make the case that XSLT is to XML what SQL is to tabular data. The relational model gets its power not from the idea of storing data in tables, but from the high-level data manipulation possible in SQL, based on the relational calculus. Equally, the hierarchic data model of XML in itself does very little to help the application developer. It is XSLT as a high-level manipulation language for XML data that provides the power.

XSLT as a language

As a language, XSLT is in some ways a rather curious beast. In this article, I won't try to give the rationale for the design decisions that were made, though they can all be traced quite logically to the requirements that the language designers identified. For a more complete exposition, see Chapter 1 of my book.

Some of the key features of the XSLT language are outlined below.

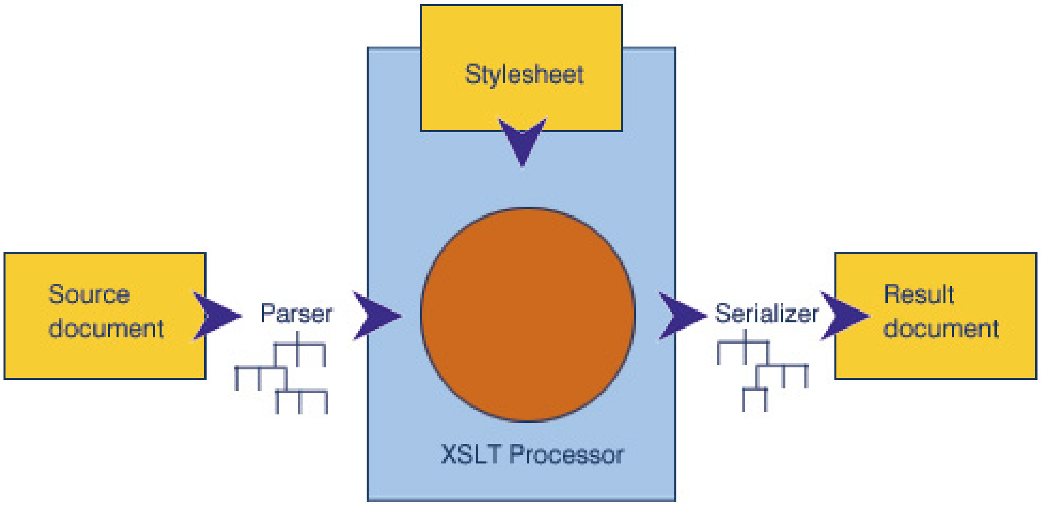
An XSLT style sheet is an XML document. The structure of the document is represented using the angle-bracket tag syntax of XML. At one level this syntax is rather clumsy, and the decision has the effect of making the language rather verbose. It does have benefits, however. It means that all the lexical apparatus of XML (for example Unicode character encoding and escaping, use of external entities, and so on) is available automatically. It means that it is easy to make an XSLT style sheet the input or output of a transformation, giving the language a reflexive capability. It also makes it easy to embed chunks of the desired XML output within the style sheet. In fact, many simple style sheets can be written essentially as a template for the desired output document, with occasional instructions embedded in the text to insert variable data from the input, or to compute a value. This makes XSLT, at this simple level, very similar to many existing proprietary HTML template languages.

The basic processing paradigm is pattern matching. In this respect XSLT inherits -- from a long tradition of text-processing languages such as Perl -- a tradition that can be traced back to languages such as SNOBOL in the 1960s. An XSLT style sheet consists of a set of template rules, each of which takes the form "if this condition is encountered in the input, then generate the following output." The order of the rules is immaterial, and there is a conflict-resolution algorithm applied when several rules match the same input. One respect in which XSLT differs from serial text processing languages, however, is that the input is not processed sequentially line by line. Rather, the input XML document is treated as a tree structure, and each template rule is applied to a node in the tree. The template rule itself can decide which nodes to process next, so the input is not necessarily scanned in its original document order.

Operation of an XSLT processor

An XSLT processor takes a tree structure as its input, and generates another tree structure as its output. This is shown in Figure 1.

Figure 1. The tree structure of XSLT input and output



The input tree structure will often be produced by parsing an XML document, and the output tree structure will often be serialized into another XML document. But the XSLT processor itself manipulates tree structures, not XML character streams. This concept, which initially strikes many users as rather academic, turns out to be crucial to understanding how to perform the more complex transformations. First, it means that distinctions in the source document that are irrelevant to the tree structure are not accessible to the XSLT processor. For example, it is not possible to apply different processing depending on whether attributes were enclosed in single quotes or double quotes, because these are considered to be different representations of the same underlying document. More subtly, it means that processing an input element, or generating an output element, is an atomic operation. It is not possible to separate the processing of the element's start tag and end tag into separate operations, because an element is represented atomically as a single node in the tree model.

XSLT uses a sublanguage called XPath to refer to nodes in the input tree. XPath is essentially a query language matched to XML's hierarchic data model. It is capable of selecting nodes by navigating the tree in any direction, and applying predicates based on the value and position of the node. It also includes facilities for basic string manipulation, numeric computation, and Boolean algebra. For example, The XPath expression `../@title` selects the title attribute of the element that is the parent of the current node. XPath expressions are used to select input nodes for processing, to test conditions during conditional processing, and to calculate values for insertion into the result tree. A simplified form of XPath expression, the pattern, is also used in template rules to define which nodes a particular template rule applies to. XPath is defined in a separate W3C Recommendation, to allow the query language to be reused in other contexts, notably the XPointer syntax for defining extended hyperlinks.

XSLT is based on the concepts of functional programming in the tradition of languages such as Lisp, Haskell, and Scheme. A style sheet is made up of templates that are essentially pure functions -- each template defines a fragment of the output tree as a function of a fragment of the input tree, and produces no side effects. The no-side-effects rule is applied quite strictly (with the exception of escapes into external code written in languages such as Java). The XSLT language allows variables to be defined, but does not allow an existing variable to change its value -- there is no assignment statement. The reason for this policy, which many new users find bewildering, is to allow style sheets to be applied incrementally. The theory is that if the language is free of side-effects, then when a small change is made to an input document it should be possible to compute the resulting change to the output document without performing the entire transformation from scratch. It has to be said that for the moment, this remains a theoretical possibility, it is not something that any existing XSLT processor achieves. (Note: Although XSLT is based on functional programming ideas, it is not as yet a full functional programming language, as it lacks the ability to treat functions as a first-class data type.)

An example style sheet
At this stage the language will become far clearer with an example. Listing 2 shows a simple style sheet to list the soccer results.

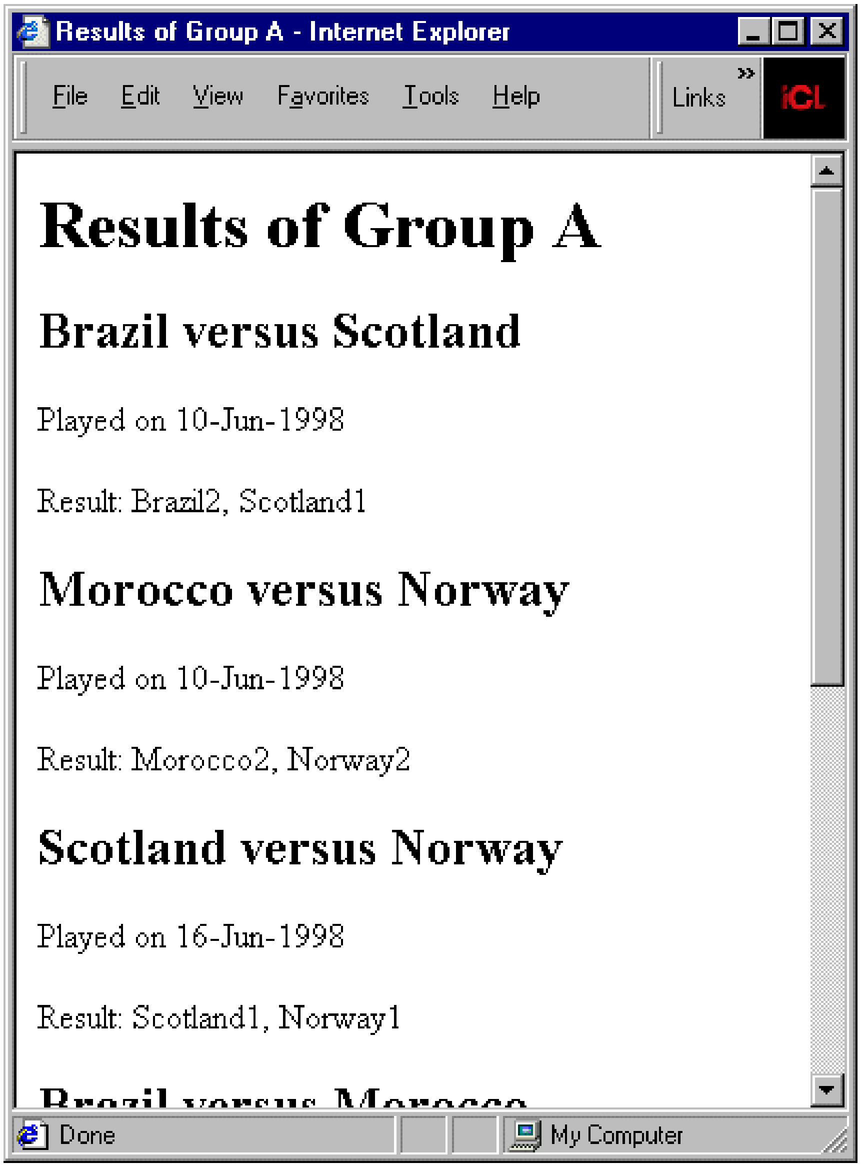
Listing 2. A basic style sheet for the soccer results

```
<xsl:transform
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="results">
    <html>
    <head><title>
        Results of Group <xsl:value-of select="@group"/>
    </title></head>
    <body><h1>
        Results of Group <xsl:value-of select="@group"/>
    </h1>
    <xsl:apply-templates/>
    </body></html>
</xsl:template>
<xsl:template match="match">
    <h2>
        <xsl:value-of select="team[1]"/> versus <xsl:value-of select="team[2]"/>
    </h2>
    <p>Played on <xsl:value-of select="date"/></p>
    <p>Result:
            <xsl:value-of select="team[1] "/>
            <xsl:value-of select="team[1]/@score"/>,
            <xsl:value-of select="team[2] "/>
            <xsl:value-of select="team[2]/@score"/>
    </p>
</xsl:template>
</xsl:transform>
```

This style sheet consists of two template rules, one to match the `<results>` element, the other to match the `<match>` element (my apologies for the homonym). The template rule for the `<results>` element outputs an HTML heading for the page, then calls `<xsl:apply-templates/>`, which is an XSLT instruction that causes all the children of the current element to be processed, each one using its appropriate template rules. In this case all the children of the `<results>` element are `<match>` elements, so they are all processed using the second template rule. The rule outputs a second-level HTML heading identifying the match (in the form "Brazil versus Scotland") and then generates HTML paragraphs giving the date of the match and the scores of both teams.

The result of this transformation is an HTML document, which is rendered by a browser as shown in Figure 2.

Figure 2. Results of the style sheet in Listing 2



Results of Group A - Internet Explorer
File Edit View Favorites Tools Help
Links
Done
My Computer

Results of Group A

Brazil versus Scotland

Played on 10-Jun-1998

Result: Brazil2, Scotland1

Morocco versus Norway

Played on 10-Jun-1998

Result: Morocco2, Norway2

Scotland versus Norway

Played on 16-Jun-1998

Result: Scotland1, Norway1

Brazil versus Morocco

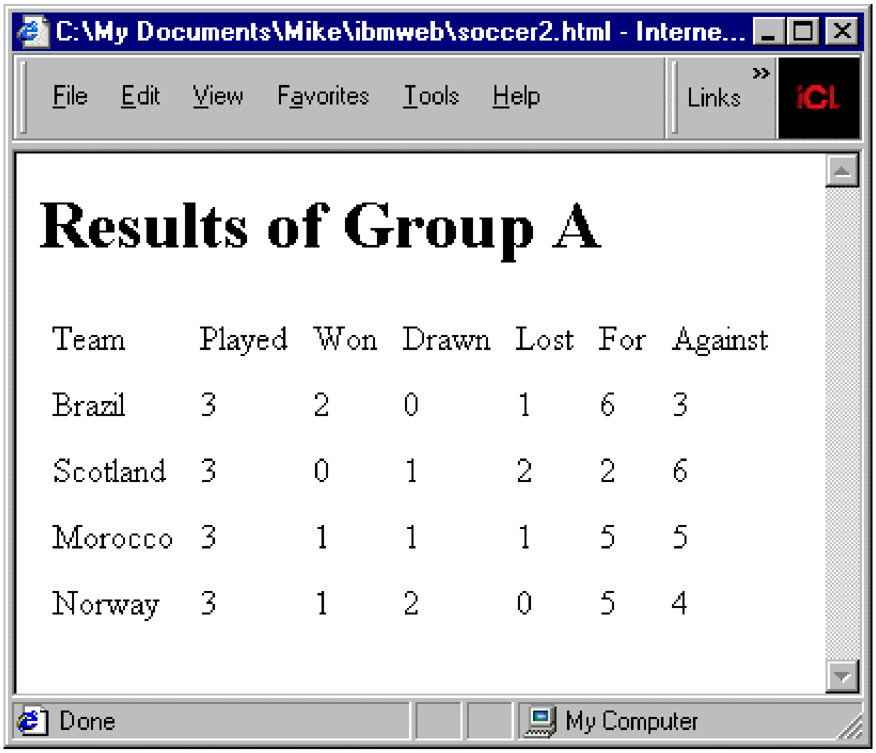
This is a very straightforward way of presenting the information. However, XSLT is much more powerful than this. Listing 3 contains another style sheet that can operate on the same source data. This time the style sheet computes a league table showing the position of the teams at the end of the tournament.

Listing 3. A style sheet that computes team standings

```
<xsl:transform
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="1.0">

<xsl:variable name="teams" select="//team[not(.=preceding::team)]"/>
<xsl:variable name="matches" select="//match"/>

<xsl:template match="results">

<html><body>
   <h1>Results of Group <xsl:value-of select="@group"/></h1>

   <table cellpadding="5">
      <tr>
        <td>Team</td>
        <td>Played</td>
        <td>Won</td>
        <td>Drawn</td>
        <td>Lost</td>
        <td>For</td>
        <td>Against</td>
      </tr>
   <xsl:for-each select="$teams">
        <xsl:variable name="this" select="."/>
        <xsl:variable name="played" select="count($matches[team=$this])"/>

        <xsl:variable name="won"
            select="count($matches[team[.=$this]/@score &gt;
team[.!=$this]/@score])"/>
        <xsl:variable name="lost"
            select="count($matches[team[.=$this]/@score &lt;
team[.!=$this]/@score])"/>
        <xsl:variable name="drawn"
            select="count($matches[team[.=$this]/@score = team[.!=$this]/@score])"/>
        <xsl:variable name="for"
            select="sum($matches/team[.=current()]/@score)"/>
        <xsl:variable name="against"
            select="sum($matches[team=current()]/team/@score) - $for"/>

        <tr>
        <td><xsl:value-of select="."/></td>
        <td><xsl:value-of select="$played"/></td>
        <td><xsl:value-of select="$won"/></td>
        <td><xsl:value-of select="$drawn"/></td>
        <td><xsl:value-of select="$lost"/></td>
        <td><xsl:value-of select="$for"/></td>
        <td><xsl:value-of select="$against"/></td>
        </tr>
   </xsl:for-each>
   </table>
</body></html>
</xsl:template>

</xsl:transform>
```

There isn't space here for a complete explanation of this style sheet, but in summary, it declares a variable for teams whose value is a node set containing one instance of each team participating in the tournament. Then for each of these teams, it calculates the total number of matches won, drawn, or lost, and the total number of goals scored for or against the team. Figure 3 shows the resulting output as it appears in the browser.

Figure 3. Results of the standings style sheet in Listing 3



Results of Group A

Team	Played	Won	Drawn	Lost	For	Against
Brazil	3	2	0	1	6	3
Scotland	3	0	1	2	2	6
Morocco	3	1	1	1	5	5
Norway	3	1	2	0	5	4

The purpose of this example is to illustrate that XSLT is capable of much more than just assigning fonts and layouts to the text that appears in the source document. It is a complete programming language that is capable of transforming the source data in arbitrary ways for presentation, or indeed for input to another application.

The benefits of XSLT

Why should you consider using XSLT?

XSLT gives you all the traditional benefits of a high-level declarative programming language, specialized to the task of transforming XML documents.

The usual benefit cited for higher-level languages is development productivity. But in truth, the real value comes from *potential for change*. An XSLT application for transforming XML data structures can be made much more resilient to changes in the details of the XML documents than a procedural application coded using the low-level DOM and SAX interfaces. In the database world this feature is known as *data independence*, and it was the quest for data independence that led to the success of declarative languages like SQL and the demise of the older navigational data access languages. I firmly believe the same will happen in the XML world.

As with all declarative languages, there is a performance penalty. But for the vast majority of applications, the performance of today's XSLT processors is already good enough to meet the application requirements, and it is getting better. In my second article, I'll discuss the kind of optimization techniques that are being used in XSLT processors such as my own Saxon product.

Summary

What I have tried to show in this article is that XSLT is a complete high-level language for manipulating XML documents in the same way that SQL is a high-level language for manipulating relational tables. It's important to understand that it's much more than just a styling language, and that it's vastly more powerful than CSS (or even CSS2).

I have already seen applications in which all the business logic is coded in XSLT. In one three-tier online banking system I looked at:

- All the data was retrieved from the back-end operational systems in the form of XML messages.
- The account data for a user was represented for the duration of an online session in the form of an XML DOM in memory.
- All information destined for the user was first packaged as an XML message, then converted to HTML by means of an XSLT transformation carried out either on the server or the client, depending on the capabilities of the browser.

The data for this application was all in XML, and the logic (including data access logic, business logic, and presentation logic) was all implemented in XSLT. I'd be frightened of recommending every project to adopt that architecture, but it has a lot going for it, and I think we'll see more systems like that in the years to come.

As a programming language, XSLT has many features -- from its use of XML syntax to its basis in functional programming theory -- that are unfamiliar to the average Web programmer. That means there is a steep learning curve and often a lot of frustration. The same was true of SQL in the early days, and all that this really proves is that XSLT is radically different from most things that have gone before. But don't give up: It's an immensely powerful technology, and well worth the effort of learning.

Resources

- The Wrox book XSLT Programmer's Reference, also by the same author. A comprehensive guide to the XSLT language.
- The XSLT 1.0 Recommendation, published by the W3C. The definitive specification of the XSLT language.
- The XPath 1.0 Recommendation, published by the W3C. The definitive specification of the XPath expression syntax used within an XSLT style sheet.
- The XSL-List, a busy mailing list for all things related to XSLT, managed by MulberryTech, with searchable archives.
- www.xslinfo.com, a good hub page with links to XSLT resources, including software, books, tutorials, and more.

About the author

Michael Kay is well known in the XML world as the author of the Saxon XSLT processor and the Wrox book *XSLT Programmer's Reference*. His background (and PhD, a long time ago) is in database technology. Since then he has designed Codasyl, relational, object-oriented, and free-text database software. Michael is also the editor of the XSLT 2.0 Working Draft, published in December 2001. In February 2001, after a 24-year stint with ICL, the UK-based IT services company, Michael moved to Software AG, where he is part of the architecture team steering the direction of future XML products. The author chose this photograph to prove that he's not always as serious as he looks on the cover of the Wrox book.

PDF e-mail it!

What do you think of this article?

○ Killer! (5) ○ Good stuff (4) ○ So-so; not bad (3) ○ Needs work (2) ○ Lame! (1)

Comments?

Submit feedback

About IBM | Privacy | Legal | Contact