



Dates are an important area in any database application

Shedding a Little Light on XML

This month is a veritable potpourri of FAQs regarding XML. I've selected some questions that people have asked me time and time again. If there's one thing that I've learned as head geek, it's that if a single person e-mails me a question, there are probably a thousand others out there scratching their heads with the same question. Well, scratch no further. If I don't cover the question this month, it'll be answered in an upcoming column.

Q: *How do I handle dates in XSLT?*

A: Although dates are nicely addressed in the upcoming 2.0 spec of XSLT (patience, grasshopper), they're noticeably absent in the 1.0 working draft. For those of you who have wrestled with date handling, I expect to hear a collective "Whoo Hoo!" (spoken like Homer Simpson) as you read this section. Nonetheless, the need for functionality to handle dates would be nice to have right now. Dates are an important area in any database application. We all use them, but handling dates in XSLT is an uncertain process.

We've all become accustomed to working with programming languages that demonstrate a robust set of functions for handling dates. Although it's very likely that you can write your own set of date functions, why reinvent the wheel!

The following list includes resources that contain some robust date functions, which can be implemented directly into your XSLT files. These functions, templates, and techniques are selflessly donated by fellow programmers who wish to lighten the burden for the rest of us by shedding some light on the subject.

Checking them out prior to embarking on an attempt to write your own might save some time and generate some ideas of your own. Here are a few examples.

- **Date sorting on a fixed date format**

Another approach is the use of substring functions to custom-tailor the sort order of date/time values. Substring functions give a pure XSLT implementation mixed with the flexibility to hit just about any section of XML input. Judging by the samples below, it becomes apparent that this technique can be extended to date/times of virtually any fixed format (yyyy/mm/dd, dd/mm/yyyy, hh:mm:ss).

In the "yyyy/mm/dd" example, we're first sorting on the year, then the month, then the day, so that we get the correct numerical sort. The other examples ("dd/mm/yyyy" and "hh:mm:ss") are variations of the principle (see Listing 1).

- **Javascript date function library**

If you search the Net long and hard, you will find code submitted by programmers dedicated to furthering the XML/XSLT cause. One great example is a collection of Javascript date handling routines written by Kurt Cagle. These include `convertToJSDate`, `AddInterval`, `DayOfWeekName`, `DayOfWeek`, `Add-`

`Interval` and `DateDiff`. All of which can be found at none other than www.kurtcagle.net.

- **Date sorting on a translated date**

The `translate()` function can be used in conjunction with substring functions to produce a concise, single-line date translation that can be used to create your date sort key. As with Example 1, this technique can also be extended to allow converting input into a numerical date sort key, even if it started out in a different format.

The following example takes a date format of "2001-01-25T10:05:53 PM" and converts it to: "20010125," which is then sorted numerically.

```
<xsl:sort select="translate
(substring-before(date,'T'), '-','')
order="descending" data-type="number" />
```

- **Advanced date sorting: sorting by month names**

Last, but not least, if you need to sort by a date that includes the name of the month, an advanced date sorting technique was contributed to Snippet-Central.com by Dimitre Novatchev. In addition to showing how dates can be sorted by month names, this snippet also demonstrates the use of the Muenchen method as well as using (simulated) arrays in XSLT. Listing 1 shows how this technique is implemented.

Note: The above solutions were selflessly contributed, along with imple-

AUTHOR BIO

David Silverlight is the chief XML evangelist for Infoteria Corporation (www.infoteria.com). He can be found at his Web site in the evening (www.XMLPitstop.com).



This ?? is an ?
for me to ? to ??? ? ????? that tug
???????????? and ??????????



mentations to demonstrate them, by the following members of our development community:

- Jason Patterson, www.vbxml.com/snippetcentral/main.asp?view=view_snippet&id=v2001022713320 ("Sorting on Dates and Other Fixed Formats Using the Substring Function")
- Kurt Cagle, www.kurtcagle.net ("Date Routines")
- Trace Wilson, www.vbxml.com/snippetcentral/main.asp?view=view_snippet&id=v20010125215939 ("Sorting by Date without Using a Schema")
- Dimitre Novatchev, www.vbxml.com/snippetcentral/main.asp?view=view_snippet&id=v20010131110050 ("Date Sorting [Advanced] Combined with the Muenchian Method for Grouping")

Q: *Is it possible to convert a fixed-length text file to XML?*

A: In an ideal world, we'll always receive our input file in the format that's easiest to work with, but in the real world we have to work with what we get. In the event that the input is a text file, it'll need to be handled a tad differently. But with a bit of effort, you can convert that text file into XML in short order.

In this example, the input file is a text file named Customers.txt, which contains one or more rows of input that is 72 characters wide. The algorithm that we'll use is as follows:

1. Reference the text file, Customers.txt, as a parsed entity. The end result will be that the element named "allcustomers" will contain the contents of the text file.

```
FixedLengthCombined.XML
<?XML version="1.0" encoding='utf-8'
?>
<!DOCTYPE allcustomers [
  <!ENTITY resourceA SYSTEM
"Customers.txt">
]>

<allcustomers>
&resourceA;
</allcustomers>
```

2. Access the "allcustomers" element and pass it to a template named "SplitFixedLengthToXML." As the name implies, this template will read through the string and create an element for all fixed-length strings contained within. At this level, the only two parameters necessary to pass to the template are:

- The contents of the string
- The length of each fixed-length string in the input file. As it processes the

block of text, it will recursively convert one line at a time into the appropriate XML elements. (See Listing 2.)

3. The inner-workings of the "SplitFixedLengthToXML" involve creating an element for all sections of the single input line that it processes. As each single line is processed, the template is called again recursively with the input text minus the text that was just processed. This process will continue until all input lines are exhausted. Listing 2 for provides the contents of the template. Notice also the use of a template, TrimTrailing (see Listing 3), to remove white space at the end of each string. This eliminates the additional spaces that by definition of a fixed-length string will very likely be at the end of each element.

Gotchas

In this solution, there are a few subtle gotchas to be aware of. First of all, text files may contain unescaped sensitive characters ("&," "<," and "[]>") that must be escaped for input to be well formed. (Yes, even though the input text is not originally in XML format, it must still be well formed to be handled by the XML processor.) No escaping = no processing. Last, in case you're wondering whether the character count is thrown off by escaping "<" into "<" – it does not. The four file bytes still represent a single character and the XML processor will handle it as such.

Cool XML Implementation of the Month

While XML specifications surface on an almost daily basis, none hits home as much as the Election Markup Language (EML). You see, I'm from Miami, so this particular specification is one that's welcomed by me personally (not that they need my endorsement). The EML proposal would include specifications for exchanging data between election and voter registration systems developed by different hardware, software, and IT services vendors. Although we missed the chance to elect the guy who invented the Internet, this is a prime example of how XML is affecting so many aspects of our lives.

On a related note, yours truly has created his own prototype of a Web-based voting system, the WeboVoteOmatic, found at www.XMLpitstop.com/WeboVoteOmatic. In reality, it just gives me an opportunity to poke fun at our political system (in a computer geek sort of way). Although it has yet to take advantage of the new EML specification, it illustrates some very useful techniques for both client-side and server-side

XML coding. Depending on the flavor of the demo that you select (source code included), it demonstrates:

1. **XMLHTTP**: Sending data between the client and server without refreshing the page (IE5+ only)
2. **Data Islands**: Binding XML data to your HTML elements
3. **SQL Server 2000**: Accessing InsertGrams, UpdateGrams, DeleteGrams
4. **BackEnd**: Saving data to SQL Server 6.5, 7.0, and 2000, Access
5. **Cross Browser**: Demonstration that works on all HTML 4+ browsers

• • •

Send questions to David at dsilverlight@infoteria.com. He'll answer questions 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**: Bring the world of wireless to your eyes

DSILVERLIGHT @ INFOTERIA.COM

LISTING 1

```
yyyy/mm/dd
<xsl:sort order="ascending" select=
"substring(text(), 1,4)" />
<xsl:sort order="ascending" select=
"substring(text(), 6,2)" />
<xsl:sort order="ascending" select=
"substring(text(), 9,2)" />
dd/mm/yyyy
<xsl:sort order="ascending" select=
"substring(text(), 7,4)" />
<xsl:sort order="ascending" select=
"substring(text(), 4,2)" />
<xsl:sort order="ascending" select=
"substring(text(), 1,2)" />
hh:mm:ss
<xsl:sort order="ascending" select=
"substring(text(), 1,2)" />
<xsl:sort order="ascending" select=
"substring(text(), 4,2)" />
<xsl:sort order="ascending" select=
"substring(text(), 7,2)" />
```

LISTING 2

```
<xsl:template match="/">
<xsl:for-each select="*">
  <xsl:variable name="allContents" select="."/>

  <!--In this template, I load the file into a
  variable so that it can be handled as a large
  string. Using the string functions like sub
  string, it is easy to parse given the start
  positions and lengths of the elements -->

  <xsl:element name="customers">
    <xsl:call-template name=
      "SplitFixedLengthToXML">
      <xsl:with-param name="strInput" select=
        "$allContents"/>
      <xsl:with-param name=
        "lineLength" select="72"/>
    </xsl:call-template>
  </xsl:element>
</xsl:for-each>
</xsl:template>
```

DOWNLOAD THE CODE @
www.xml-journal.com