

Overcoming decades old legacy systems with XSLT and Muenchian grouping

By Edmond Woychowsky

Takeaway

Implementing B2B Web services sometimes involves interaction with entrenched legacy systems. One method application developers can use in these situations takes advantage of XSLT and Muenchian grouping.

Table of Contents

OVERCOMING DECADES OLD LEGACY SYSTEMS WITH XSLT AND MUENCHIAN GROUPING	2
MUENCHIAN GROUPING	2
<i>Listing A</i>	3
<i>Listing B</i>	4
MULTIPLE CRITERIA	5
<i>Listing C</i>	5
AN ADAPTABLE SOLUTION.....	6
DESCRIPTION	6
VERSION HISTORY	6
▪ 1.0	6
TECHREPUBLIC SITE FEATURES	6

Overcoming decades old legacy systems with XSLT and Muenchian grouping

Overcoming decades old legacy systems with XSLT and Muenchian grouping

Recently I needed to create a Web service that would allow an outside organization to enter orders on a legacy system using [XML](#). My initial optimism about this project was squashed by the reality of dealing with the limitations of a system that was designed and coded sometime during the Carter Administration. While I had anticipated that the XML would need to be massaged in order to produce something that my mainframe counterpart could use, I hadn't foreseen exactly what constituted an order on both the outside organization and the legacy application.

Muenchian grouping

Most of the major philosophical differences in what constitutes an order stemmed from the fact that the outside organization stored customer information on the line level while the legacy application stored customer information on the order level. This means that every line on the inbound order could conceivably be a different customer, a concept that was not contemplated during the Carter years. Another issue which only served to compound this problem is that the legacy system only supported a single ship to address, a single bill to address and a single carrier per order. What had started out as a relatively simple Web service quickly began to snow ball into both a major project and a major headache.

After a brief delusional moment where some kind of convoluted C# program was considered for splitting an inbound order into something the legacy system could handle, I decided to try a more elegant approach, [XSLT](#) and [Muenchian grouping](#).

Developed by, and named after, [Steve Muench](#) of Oracle, the Muenchian grouping method uses the `xsl:key` element and the `key()` and `generate-id()` functions to determine keys that are used to retrieve nodes. To illustrate how this works let's start with the XML document shown in **Listing A** and, for the sake of simplicity, group upon only the customer node. With Muenchian grouping we would create XSLT that looks something like what is shown in **Listing B**.

Overcoming decades old legacy systems with XSLT and Muenchian grouping

Listing A

```
<?xml version="1.0" encoding="UTF-8"?>
<order>
  <line>
    <customer>customer 1</customer>
    <shipto street1="Northampton Street" city="Easton" state="PA" zip="18042"
country="US"/>
    <billto street1="Northampton Street" city="Easton" state="PA" zip="18042"
country="US"/>
    <carrier>UPS</carrier>
    <item>1</item>
  </line>
  <line>
    <customer>customer 2</customer>
    <shipto street1="Woodbridge Avenue" city="Edison" state="NJ" zip="08837"
country="US"/>
    <billto street1="Woodbridge Avenue" city="Edison" state="NJ" zip="08837"
country="US"/>
    <carrier>UPS</carrier>
    <item>2</item>
  </line>
  <line>
    <customer>customer 2</customer>
    <shipto street1="Woodbridge Avenue" city="Edison" state="NJ" zip="08837"
country="US"/>
    <billto street1="Woodbridge Avenue" city="Edison" state="NJ" zip="08837"
country="US"/>
    <carrier>FED</carrier>
    <item>3</item>
  </line>
  <line>
    <customer>customer 1</customer>
    <shipto street1="Butler Street" city="Easton" state="PA" zip="18042"
country="US"/>
    <billto street1="Northampton Street" city="Easton" state="PA" zip="18042"
country="US"/>
    <carrier>UPS</carrier>
    <item>4</item>
  </line>
  <line>
    <customer>customer 1</customer>
    <shipto street1="Northampton Street" city="Easton" state="PA" zip="18042"
country="US"/>
    <billto street1="Northampton Street" city="Easton" state="PA" zip="18042"
country="US"/>
    <carrier>UPS</carrier>
    <item>5</item>
  </line>
</order>
```

Copyright ©2005 CNET Networks, Inc. All rights reserved.

To see more downloads and get your free TechRepublic membership, please visit

<http://techrepublic.com.com/2001-6240-0.html>

Listing B

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:key name="keyGroup" match="line" use="customer"/>
  <xsl:template match="/">
    <xsl:element name="ordergroup">
      <xsl:apply-templates select="//line[generate-id(.) = generate-
id(key('keyGroup',customer)[1])]" mode="modeOrder"/>
    </xsl:element>
  </xsl:template>
  <xsl:template match="line" mode="modeOrder">
    <xsl:variable name="currentKey">
      <xsl:value-of select="customer"/>
    </xsl:variable>
    <xsl:element name="order">
      <xsl:apply-templates select="//line[customer = $currentKey]" mode="modeLine"/>
    </xsl:element>
  </xsl:template>
  <xsl:template match="line" mode="modeLine">
    <xsl:copy-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```

At first glance Muenchian grouping doesn't appear to make much sense, how can an *xsl:key* element along with a couple of functions group the nodes in an XML document? In order for the inner workings to become clear it is necessary to examine the individual components, only then will the simple elegance become clear.

The *xsl:key* element is a top-level XSLT element that is used to declare a named key. The element's *match* attribute defines the node returned from the *key()* function while the *use* attribute defines the key itself. The purpose of the predicate, [1], is to insure that only the first instance of each key is returned. This means that each key is used only once instead of the number of times that the key occurs. Without this predicate, if a particular key occurred twice the associated elements would be doubled, three times tripled, and so forth.

The *generate-id()* function is the final piece, it returns a string that uniquely identifies a node, regardless of how the node is obtained. So, if *generate-id(.)* is equal to *generate-id(key('keyGroup',customer)[1])* is true then both nodes are the same node, just obtained differently. It is also important to remember that while the string is always unique, there is no guarantee that *id* will be equal from execution to execution.

Overcoming decades old legacy systems with XSLT and Muenchian grouping

Multiple criteria

Occasionally, when grouping, it is necessary to group information on more than one criteria. Consider the XML document from Listing A. Wouldn't it make sense to, in addition to grouping on customer, group on carrier, ship to address, and bill to address? Seems easy enough, just add another `xsl:key` element and another `key()` function, right? Wrong!

When using Muenchian grouping only one `xsl:key` element and `key()` function is ever needed. Multiple grouping criteria is handled through the addition of the `concat()` function which is used to concatenate the various nodes and/or elements to produce a single key. With this in mind, **Listing C** shows the XSLT that performs grouping on the customer, carrier, ship to address and bill to address.

Listing C

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:key name="keyGroup" match="line"
use="concat(customer,carrier,shipto/@street1,shipto/@street2,shipto/@city,shipto/@state,shipto/@zip,billto/@street1,billto/@street2,billto/@city,billto/@state,billto/@zip)"/>
  <xsl:template match="/">
    <xsl:element name="ordergroup">
      <xsl:apply-templates select="//line[generate-id(.) = generate-id(key('keyGroup',concat(customer,carrier,shipto/@street1,shipto/@street2,shipto/@city,shipto/@state,shipto/@zip,billto/@street1,billto/@street2,billto/@city,billto/@state,billto/@zip))][1)]" mode="modeOrder"/>
    </xsl:element>
  </xsl:template>
  <xsl:template match="line" mode="modeOrder">
    <xsl:variable name="currentKey">
      <xsl:value-of
select="concat(customer,carrier,shipto/@street1,shipto/@street2,shipto/@city,shipto/@state,shipto/@zip,billto/@street1,billto/@street2,billto/@city,billto/@state,billto/@zip)"/>
    </xsl:variable>
    <xsl:element name="order">
      <xsl:apply-templates
select="//line[concat(customer,carrier,shipto/@street1,shipto/@street2,shipto/@city,shipto/@state,shipto/@zip,billto/@street1,billto/@street2,billto/@city,billto/@state,billto/@zip) = $currentKey]" mode="modeLine"/>
    </xsl:element>
  </xsl:template>
  <xsl:template match="line" mode="modeLine">
    <xsl:copy-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```

How far can Muenchian grouping using concatenated be taken? Personally I've used this technique to group information based upon eleven different criteria. The resulting XSLT was, when printed not quite three pages long, which while impressive for XSLT, was thousands of lines shorter than programs written in traditional programming languages.

Copyright ©2005 CNET Networks, Inc. All rights reserved.

To see more downloads and get your free TechRepublic membership, please visit

<http://techrepublic.com.com/2001-6240-0.html>

Overcoming decades old legacy systems with XSLT and Muenchian grouping

An adaptable solution

While this may, at first, seem like a rather specific application it is one of those tools that can be readily adapted to other uses. I recommend playing with the examples provided here in order to become comfortable with this technique. In addition, you may want to purchase a copy of XMLSPY from [Altova](#), which was used to produce and test the examples provided here. Not only is it a slick professional XML editor, it allows the developer to step-through an XSLT element by element, which comes in handy when debugging Muenchian grouping.

Description

Implementing B2B Web services sometimes involves interaction with entrenched legacy systems. While the flexibility of XML can often be enough to compensate for the limitations of these systems, it is not always the only technology required. One method application developers can use in these situations takes advantage of XSLT and Muenchian grouping. Follow the example code explained in this download to see how this sophisticated technique overcame legacy limitations. The lessons learned may help you in your next project.

Version history

- **1.0**

TechRepublic communities engage IT professionals in the ultimate peer-to-peer experience, providing actionable information, tools, and services to help members get their jobs done. TechRepublic serves the needs of the professionals representing all segments of the IT industry, offering information and tools for IT decision support and professional advice by job function.

TechRepublic site features

[Free newsletters:](#) Keep up-to-date with the IT industry with our newsletters, which cover various topics including disaster recovery, Internet security, Microsoft Office, e-mail administration, management advice, and much more.

[Free downloads:](#) We've collected resources to make your job easier, including ready-to-use IT forms and templates, checklists, tools, executables, Gartner product analyses, and white papers.

[TechRepublic's books and CDs:](#) Find the latest books and CDs about today's critical IT topics, including PC troubleshooting, VPN, TCP/IP, Windows client and server issues, and Cisco administration.

[Discussion center:](#) Open a discussion thread on any article or column or jump into preselected topics: career, technology, management, and miscellaneous. The fully searchable Discussion Center brings you the hottest discussions and threads and allows you to sort them by topic. Our online IT community provides real-world solutions and the latest articles, resources, and discussions affecting frontline IT pros. Get access to more than 250 full-text IT books, along with exclusive downloads and in-depth articles on network and system administration, PC troubleshooting, help desk and support issues, and more.

Copyright ©2005 CNET Networks, Inc. All rights reserved.

To see more downloads and get your free TechRepublic membership, please visit

<http://techrepublic.com.com/2001-6240-0.html>