Building XQuery with Stylus Studio

Web Service Aggregation and Reporting

sonic
SOFTWARE™

Contents

Introduction

The widespread adoption of XML has profoundly altered the way that information is exchanged within and between enterprises. XML, an extensible, text-based markup language, describes data in a way that is both hardware- and software-independent. As such, it has become a standard of choice for a growing number of Web services and Service Oriented Architectures. With a vast amount of data being published in XML format by multiple sources, the need has arisen for an easy and efficient means of extracting and manipulating this information. XQuery has emerged as an ideal way to aggregate data from Web services, relational databases, and other applications that employ XML.

This article illustrates a scenario in which XQuery is used for aggregating data from multiple sources, enabling reporting tools such as XSLT to present that information in HTML or other formats as desired. Sonic Stylus Studio 5.0 is the core technology used in developing this scenario. Sonic Stylus Studio is an award-winning XML development environment that includes an XSLT editor and debugger, a graphical XML Schema designer, an XML-to-XML mapper, a WYSIWYG XML-to-HTML designer, and an advanced XQuery editor and debugger.

Scenario

To illustrate the use of XQuery for aggregating data from multiple sources, we'll be using real-world data from a stock-quote Web service, and combining that information with historical company data stored in a relational database and presented as XML. In this example, the historical data is being enhanced with live data about the current stock price, which is being retrieved via a Web-service call. Once the data is aggregated, it can be presented in any number of formats. For the purpose of this article, we'll use Sonic Stylus Studio to display the data in HTML using XSLT.

Figure 1 shows an end-to-end overview of the scenario. We'll work through the various stages in more detail throughout this article.

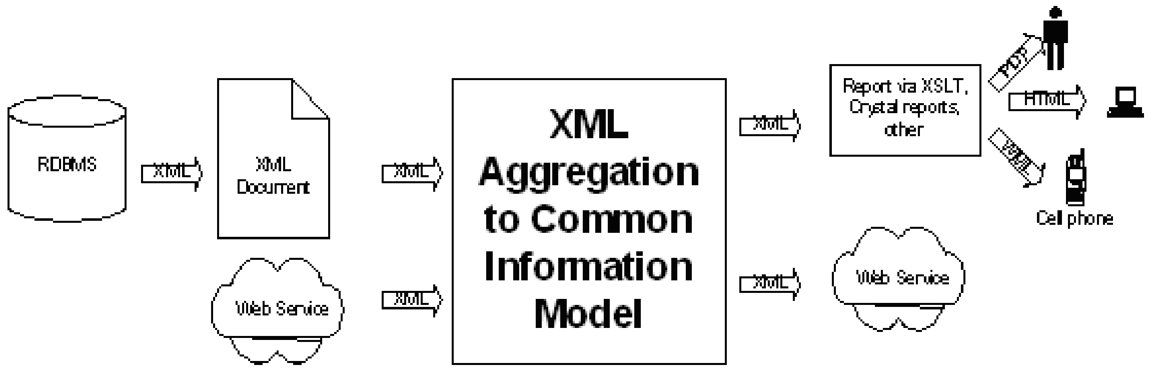
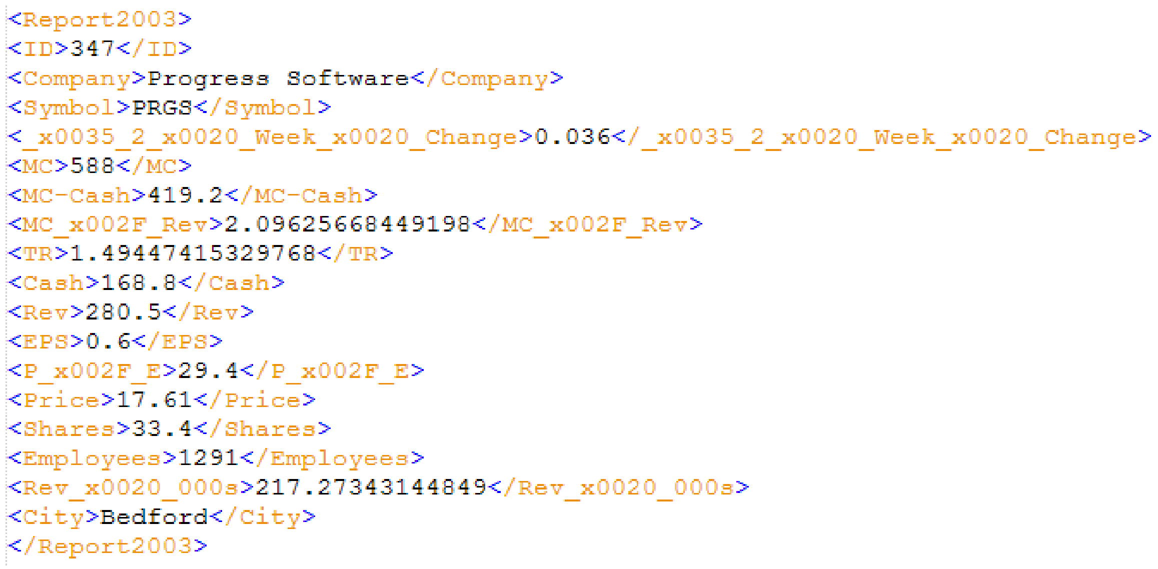

RDBMS
XML
XML Document
XML
Web Service
XML
XML Aggregation to Common Information Model
XML
XML
Report via XSLT, Crystal reports, other
PDF
HTML
WML
Cell phone
Web Service

Figure 1: Using XML for data aggregation and reporting

There are two XML inputs in this example: the relational database (RDBMS) and the Web service. For the RDBMS, Microsoft Access is being used to automatically format the data into XML. This conversion can also be performed by Sonic Stylus Studio's built-in ADO-to-XML Document Wizard. An excerpt of the converted data from the RDBMS is seen in Figure 2.

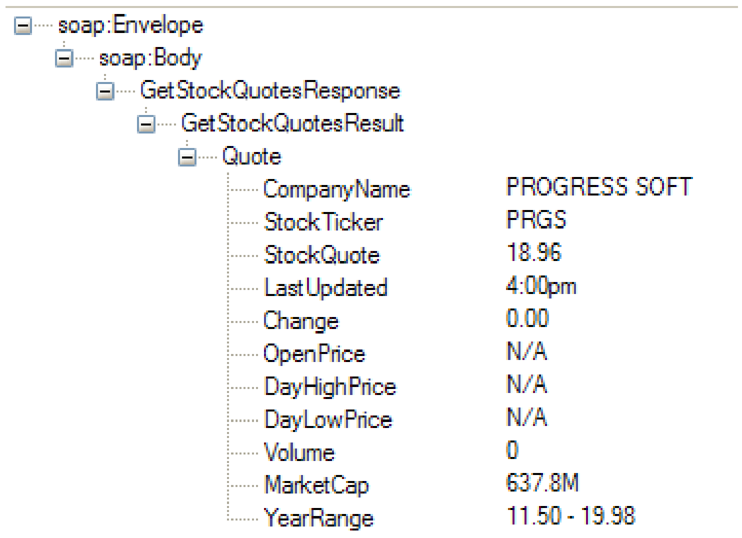


```
<Report2003>
<ID>347</ID>
<Company>Progress Software</Company>
<Symbol>PRGS</Symbol>
<_x0035_2_x0020_Week_x0020_Change>0.036</_x0035_2_x0020_Week_x0020_Change>
<MC>588</MC>
<MC-Cash>419.2</MC-Cash>
<MC_x002F_Rev>2.09625668449198</MC_x002F_Rev>
<TR>1.49447415329768</TR>
<Cash>168.8</Cash>
<Rev>280.5</Rev>
<EPS>0.6</EPS>
<P_x002F_E>29.4</P_x002F_E>
<Price>17.61</Price>
<Shares>33.4</Shares>
<Employees>1291</Employees>
<Rev_x0020_000s>217.27343144849</Rev_x0020_000s>
<City>Bedford</City>
</Report2003>
```

Figure 2: Historical stock data retrieved from Access database

The other source of XML data in this example is an actual real-time stock-quote Web service, which is located at:
http://www.swanandmokashi.com/HomePage/WebServices/StockQuotes.asmx?WSDL#StockQuotesSoap

The result of executing this Web service is a SOAP response that can be seen in Figure 3. SOAP (Simple Object Access Protocol) is the messaging protocol that allows Web service applications to talk to each other. When a Web service is identified as an input source, Sonic Stylus Studio transparently invokes the Web service as part of any operation in which that data source is being used.



```
soap:Envelope
  soap:Body
    GetStockQuotesResponse
      GetStockQuotesResult
        Quote
          CompanyName      PROGRESS SOFT
          StockTicker      PRGS
          StockQuote       18.96
          LastUpdated      4:00pm
          Change           0.00
          OpenPrice        N/A
          DayHighPrice     N/A
          DayLowPrice      N/A
          Volume           0
          MarketCap        637.8M
          YearRange        11.50 - 19.98
```

Figure 3: Sample SOAP response from stock-price Web service

In order to extract and aggregate the XML data from both the Microsoft Access database and the Web service, we'll build a common information model in XML. This will enable us to more easily manipulate the results of the data aggregation for reporting purposes. Separating presentation from data extraction enables us to develop reporting algorithms independently, which makes for a more easily maintainable system. The schema for the common information model as used for the stock data results is depicted in Figure 4.

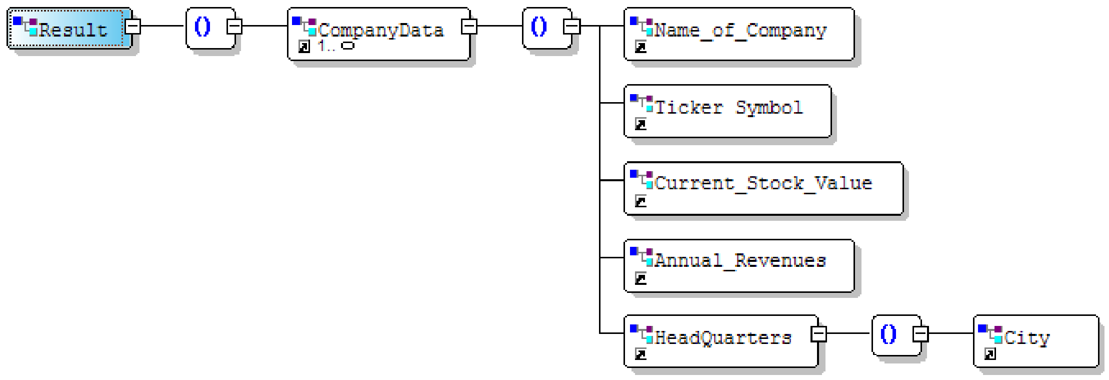


Figure 4: Common information model for stock data aggregation

Writing the solution in XQuery

Because XQuery was designed for processing XML, it's a logical choice for this scenario. Furthermore, XQuery easily supports the notion of "joins," which allows for two or more data sources to be combined based on query conditions. Other language options include:

- Using a high-level language like Java and performing the parsing and manipulation of the data using JDOM
- Using XSLT

The major difference is that with XSLT, the join logic is accomplished by saving the stock symbol in a variable in an outer loop when processing the SOAP stock-ticker information, and then in an inner loop testing that variable in an "xsl:if" against each stock element of the historical stock data. When the "xsl:if" evaluates to true, Annual_Revenues and City are output.

For the XQuery solution, we first build a cross-product of the elements from the two XML files, and then limit the result by returning only those items from both the Access database and SOAP sources that match. The complete program is shown below:

```
declare namespace soap = "http://schemas.xmlsoap.org/soap/envelope/"
declare namespace a = "http://swanandmokashi.com/"

<Result>
{
      for $Quote in
      /soap:Envelope/soap:Body/a:GetStockQuotesResponse/a:GetStockQuote
      sResult/a:Quote,
       $Report2003 in document("Report2003.xml")/dataroot/Report2003
      where $Report2003/Symbol = $Quote/a:StockTicker
      return
            <CompanyData>
                  <Name_of_Company>
                        {$Report2003/Company/text()}
                  </Name_of_Company>
                  <Ticker_Symbol>
                        {$Report2003/Symbol/text()}
                  </Ticker_Symbol>
```

```
                    <Current_Stock_Value>
                          {$Quote/a:StockQuote/text()}
                    </Current_Stock_Value>
                    <Annual_Revenues>
                          {$Report2003/Rev/text()}
                    </Annual_Revenues>
                    <HeadQuarters>
                          <City>
                                {$Report2003/City/text()}
                          </City>
                    </HeadQuarters>
              </CompanyData>
}
</Result>
```

Let’s walk through the program line by line. The first two lines of the program are namespace declarations, which are used for accessing the data contained within a SOAP document.

```
 declare namespace soap = "http://schemas.xmlsoap.org/soap/envelope/"
 declare namespace a = http://swanandmokashi.com/
```

These declarations are required because SOAP requests contain two distinct namespaces: one that describes the structure of the message, and one that describes the payload of the message – which, in this case, is the real-time stock information. The two namespaces prevent naming conflicts. The elements referring to the SOAP components of the message are accessed with a “soap:” prefix (/soap:Envelope/soap:Body), whereas elements that are part of the embedded message that is transferred as part of the Web service response use the “a:” prefix ($Quote/a:StockTicker).

The next two lines provide a clue as to how the program is going to operate:

```
 <Result>
 {
```

When the XQuery starts to execute, it builds an XML tree starting with the <Result> element. The squiggly bracket “{” that follows is the start of a processing block that, when completed, will have its output added to the XML tree.

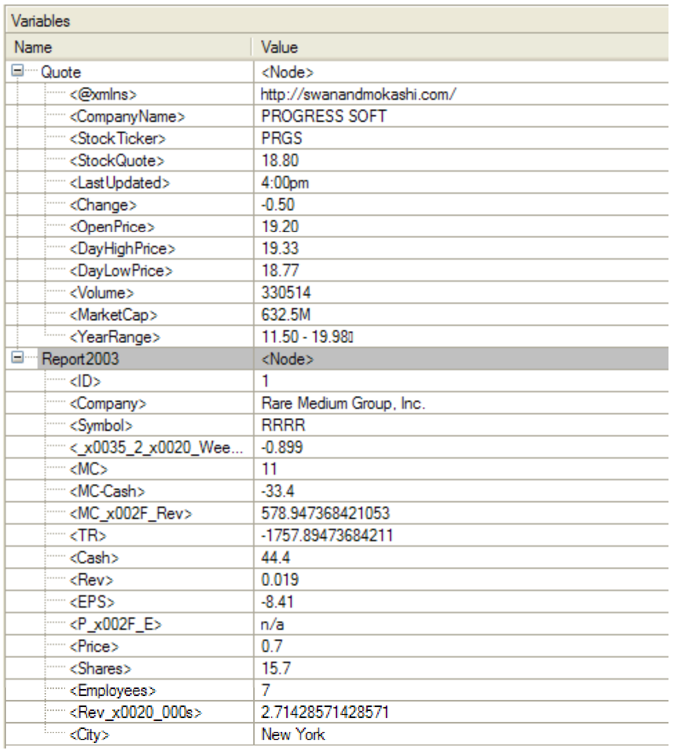
The bulk of the processing of the program occurs in the “for/where” loop.

```
 for $Quote in
/soap:Envelope/soap:Body/a:GetStockQuotesResponse/a:GetStockQuotesResul
t/a:Quote,
 $Report2003 in document(“Report2003.xml”)/dataroot/Report2003
 where $Quote/a:StockTicker = $Report2003/Symbol
```

The “for” loop iterates over the SOAP message over the “a:Quote” repeating element. Each time through the loop, it assigns the “a:Quote” element to the “$Quote” variable. Similarly, for each of the “Report2003” repeating elements from the “Report2003.xml” file, the program assigns the element to $Report2003. Each time through the loop, the “where” clause executes and when it evaluates to true, it lets the next line of the XQuery run.

Note that an XQuery processor can in many cases optimize the execution of a query such as this. For instance, it can take the result of the “$Quote” variable and use that to more optimally read from the “Report2003.xml” document, creating or using an existing index if necessary. The XQuery processor could also read the “Report2003.xml” document just once and create an in-memory hash-table of the structure so that during the next iteration through the loop, the “$Quote/a:StockTicker = $Report2003/Symbol” comparison can be done very efficiently.

Let’s take a look at the state of the “$Quote” and “$Report2003” variables after one run through the loop:


Variables

Name	Value
Quote	<Node>
<@xmlns>	http://swanandmokashi.com/
<CompanyName>	PROGRESS SOFT
<StockTicker>	PRGS
<StockQuote>	18.80
<LastUpdated>	4:00pm
<Change>	-0.50
<OpenPrice>	19.20
<DayHighPrice>	19.33
<DayLowPrice>	18.77
<Volume>	330514
<MarketCap>	632.5M
<YearRange>	11.50 - 19.98
Report2003	<Node>
<ID>	1
<Company>	Rare Medium Group, Inc.
<Symbol>	RRRR
<_x0035_2_x0020_Wee...	-0.899
<MC>	11
<MC-Cash>	-33.4
<MC_x002F_Rev>	578.947368421053
<TR>	-1757.89473684211
<Cash>	44.4
<Rev>	0.019
<EPS>	-8.41
<P_x002F_E>	n/a
<Price>	0.7
<Shares>	15.7
<Employees>	7
<Rev_x0020_000s>	2.71428571428571
<City>	New York

Continuing through the code, once a “where” condition is satisfied, the return clause runs. Here’s what the first few lines of the return look like:

```
return
      <CompanyData>
            <Name_of_Company>
                  {$Report2003/Company/text()}
            </Name_of_Company>
```

Here, an XML result tree is being built as part of the return operation, with <CompanyData> at the top-level, and <Name_of_Company> as a nested sub-element. The value being evaluated by the query is defined within the squiggly brackets. Note that you must specify the datatype being retrieved [here, it is “text ()”]; otherwise, the XQuery copies just the “$Report2003/Company” element – and not the contents – to the destination XML result tree.

As part of Sonic Stylus Studio’s XSLT debugger, the intermediate result of returns is made available in the product’s “Variables” window. This is especially helpful for developing XQuery programs because, unlike with languages like XSLT, the output isn’t generated immediately. Therefore, this tool allows for quickly looking at the intermediate state during execution.

Skipping past the rest of the elements in the return statement, we see the following:

```
}
</Result>
```

The end squiggly bracket "}" takes the XML result tree that was generated (starting with the begin squiggly bracket "{") and adds it to the outer XML result tree. Finally, the end tag of the <Result> is added to the XML result tree and the XQuery program completes.

Achieving the result

When the XQuery is executed within Sonic Stylus Studio, the XML result tree is output to the Stylus Preview Window. From there, sorting the result according to any element from the input can be useful, and creating an extension to accomplish that is an easy matter. To sort the result in alphabetical order by company name, for example, simply insert an "order by" statement after the "where" in the "for/let/where/return" loop. We can then sort the output by company name with the following loop:

```
for $Quote in
/soap:Envelope/soap:Body/a:GetStockQuotesResponse/a:GetStockQuotesResul
t/a:Quote,
 $Report2003 in document("Report2003.xml")/dataroot/Report2003
where $Quote/a:StockTicker = $Report2003/Symbol
order by $Quote/a:CompanyName
return
…
```

While the code to produce the above XQuery is fairly straightforward to write, Sonic Stylus Studio provides a mapping tool that simplifies the building of basic maps. Building the XQuery as demonstrated in this article can be accomplished in less than a minute using this mapping tool.

The visual representation of the XQuery is seen in Figure 5. The "for/let/where" loop is constructed by dragging the repeating elements to a FLWOR icon (where the "for" clause is implemented); the "where" clause is implemented by creating an "equal" icon and dragging the elements for both sides of the "equal" to that icon. The "for/let/where" loop is completed by hooking the "equal" to the FLWOR and then dragging the output of the FLWOR to the repeating element of the target document. After creating the structure of the "for/let/where" loop, elements from the source schema can simply be dragged and dropped onto the target schema.

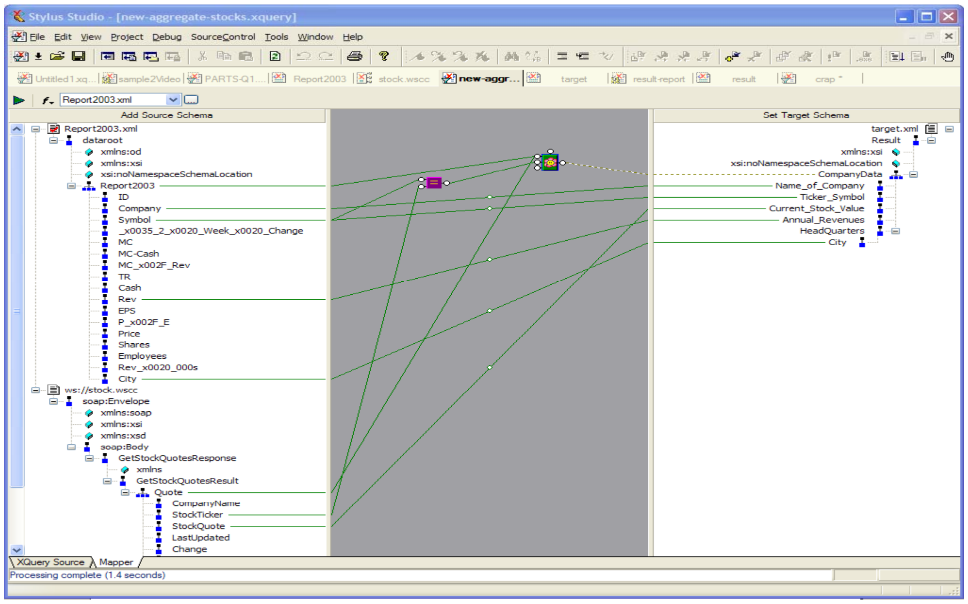
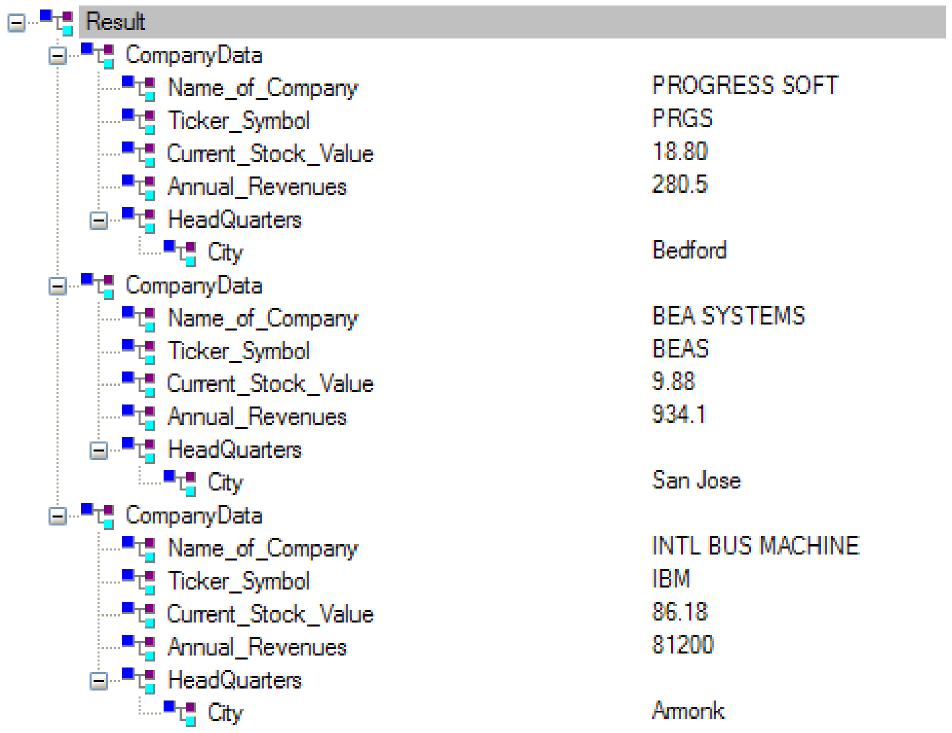


Figure 5: Sonic Stylus Studio's XQuery Visual Mapper

Figure 6 shows a sample of resulting data produced by this XQuery:



```
Result
  CompanyData
    Name_of_Company          PROGRESS SOFT
    Ticker_Symbol            PRGS
    Current_Stock_Value      18.80
    Annual_Revenues          280.5
    HeadQuarters
      City                   Bedford
  CompanyData
    Name_of_Company          BEA SYSTEMS
    Ticker_Symbol            BEAS
    Current_Stock_Value      9.88
    Annual_Revenues          934.1
    HeadQuarters
      City                   San Jose
  CompanyData
    Name_of_Company          INTL BUS MACHINE
    Ticker_Symbol            IBM
    Current_Stock_Value      86.18
    Annual_Revenues          81200
    HeadQuarters
      City                   Armonk
```

Figure 6: Result of Xquery-based data aggregation

Reporting to HTML using XSLT

Once the XML-based common information model is populated, there are many options for presenting that information to the user. To complete this example, we'll create a style-sheet using Sonic Stylus Studio's WYSIWYG XML-to-HTML designer to present the data as HTML. Other options include using a reporting tool such as Crystal Reports, or processing the XML with another language such as Visual Basic or Java and then writing custom reports from that. Another option, of course, is simply to use XQuery to generate HTML.

This was generated by Sonic Stylus Studio:

Sample Stock Report

Company Name	Ticker	Stock Price	Annual Revenue	HeadQuarters
Progress Software	PRGS	19.29	280.5	Bedford
Int'l Business Machines Corp	IBM	86.32	81200	Armonk
BEA Systems, Inc.	BEAS	10.29	934.1	San Jose

Summary

Web services provide a wealth of new information that is described and made available to applications in XML. Often, data analysis requires information from multiple sources, which means that Web service data, for example, needs to be enhanced or aggregated with XML data obtained from other data sources.

Using Sonic Stylus Studio, we proved it to be quick and easy to create an XQuery that efficiently aggregates data from historical stock data stored in a relational database with live stock-quote data provided from a Web service. A common information model was used for the target of the aggregation, and the result was translated into HTML for presentation purposes.

Other Resources

All of the sources used in this article can be found in www.StylusStudio.com/articles/stock-aggregation.zip.

The tool used for creation of this article is Sonic Stylus Studio. It can be downloaded from http://www.sonicsoftware.com/products/additional_software/stylus_studio/index.ssp.

A good introductory article on XQuery can be found at:
http://www.xml.com/pub/a/2002/10/16/xquery.html.

The World Wide Web Consortium website use-case document has numerous and useful examples on the use of XQuery. See http://www.w3.org/TR/xquery-use-cases/.