Tìm hiểu cấu trúc và cú pháp của XML

Để thấy ảnh hưởng rộng lớn của **XML** trong ngành Công Nghệ Thông Tin cận đại bạn chỉ cần để ý rằng XML là lý do của sự hiện hữu (raison d'être) của Microsoft .Net. Từ WindowsXP trở đi, bên trong đầy dẫy XML. Microsoft đã đầu tư hơn 3 tỷ đô la Mỹ vào kỹ thuật nầy, và trong tương lai gần đây tất cả phần mềm của Microsoft nếu không dọn nhà (được ported) qua .NET thì ít nhất cũng được .NET Enabled (dùng cho .NET được). Đi song song với .NET là **SQLServer 2000**, một cơ sở dữ liệu hổ trợ XML hoàn toàn.

Có lẽ bạn đã nghe qua **Web Services**. Đó là những dịch vụ trên Web ta có thể dùng on-demand , tức là khi nào cần cho chương trình của mình, bằng cách gọi nó theo phương pháp giống giống như gọi một Hàm (Function). Web Services được triển khai dựa vào XML và Http, chuẩn dùng để gởi các trang Web.

Điểm quan trọng của kỹ thuật XML là nó không thuộc riêng về một công ty nào, nhưng là một tiêu chuẩn được mọi người công nhận vì được soạn ra bởi World Wide Web Consortium - W3C (một ban soạn thảo với sự hiện diện của tất cả các dân có máu mặt trên giang hồ Tin học) và những ai muốn đóng góp bằng cách trao đổi qua Email. Bản thân của XML tuy không có gì khó hiểu, nhưng các công cụ chuẩn được định ra để làm việc với XML như **Document Object Model - DOM, XPath, XSL**, v.v.. thì rất hữu hiệu, và chính các chuẩn nầy được phát triển không ngừng.

Microsoft committed (nhất quyết dấn thân) vào XML ngay từ đầu. Chẳng những có đại diện để làm việc thường trực trong W3C mà còn tích cực đóng góp bằng cách gởi những đề nghị. Vị trí của Microsoft về XML là khi tiêu chuẩn chưa được hoàn thành thì các sản phẩm của Microsoft tuân thủ (comply) những gì có vẽ được đa số công nhận và khi tiêu chuẩn hoàn thành thì tuân thủ hoàn toàn.

Cái công cụ XML sáng giá nhất của Microsoft là ActiveX **MSXML**. Nó được dùng trong Visual Basic 6, ASP (Active Server Pages) của IIS và Internet Explorer từ version 5.5. Hiện nay MSXML đã có version 4.0. MSXML parse (đọc và phân tích) và validate (kiểm tra sự hợp lệ) XML file để cho ta DOM, một tree của các Nodes đại diện các thành phần bên trong XML. MSXML cũng giúp ta dựa vào một XSL file để transform (biến thể) một XML file thành một trang Web (HTML) hay một XML khác.

XML là gì?

Một chút lịch sử

Như tất cả chúng ta đều biết, **XML** là viết tắt cho chữ **eXtensible Markup Language** - nhưng Markup Language (ngôn ngữ đánh dấu) là gì?

Trong ngành ấn loát, để chỉ thị cho thợ sắp chữ về cách in một bài vỡ, tác giả hay chủ bút thường vẽ các vòng tròn trong bản thảo và chú thích bằng một ngôn ngữ đánh dấu tương tự như tốc ký. Ngôn ngữ ấy được gọi là **Markup Language**.

XML là một ngôn ngữ đánh dấu tương đối mới vì nó là một subset (một phần nhỏ hơn) của và đến từ (derived from) một ngôn ngữ đánh dấu già dặn tên là **Standard Generalized Markup Language (SGML)**. Ngôn ngữ HTML cũng dựa vào SGML, thật ra nó là một áp dụng của SGML.

SGML được phát minh bởi Ed Mosher, Ray Lorie và Charles F. Goldfarb của nhóm IBM research vào năm 1969, khi con người đặt chân lên mặt trăng. Lúc đầu nó có tên là **Generalized Markup Language (GML)**, và được thiết kế để dùng làm **meta-language**, một ngôn ngữ được dùng để diễn tả các ngôn ngữ khác - văn phạm, ngữ vựng của chúng ,.v.v.. Năm 1986, SGML được cơ quan **ISO (International Standard Organisation)** thu nhận (adopted) làm tiêu chuẩn để lưu trữ và trao đổi dữ liệu. Khi Tim Berners-Lee triển khai **HyperText Markup Language - HTML** để dùng cho các trang Web hồi đầu thập niên 1990, ông ta cứ nhắc nhở rằng HTML là một áp dụng của SGML.

Vì SGML rất rắc rối, và HTML có nhiều giới hạn nên năm 1996 tổ chức W3C thiết kế XML. XML version 1.0 được

định nghĩa trong hồ sơ **February 1998 W3C Recommendation**, giống như một **Internet Request for Comments (RFC)**, là một **"tiêu chuẩn"**.

Từ HTML đến XML

Trong một trang Web, ngôn ngữ đánh dấu HTML dùng các cặp **Tags** để đánh dấu vị trí đầu và cuối của các mảnh dữ liệu để giúp chương trình trình duyệt (browser) parse (ngắt khúc để phân tích) trang Web và hiển thị các phần theo ý người thiết kế trang Web. Thí dụ như một câu HTML dưới đây:

```
<P align="center">Chào mừng bạn đến thăm
  <STRONG>Vovisoft</STRONG>Web site
</P>
```

Câu code HTML trên có chứa hai markup Tags, **<P>** và **<STRONG>**. Mỗi cặp Tags gói dữ liệu nó đánh dấu giữa **opening Tag** và **closing Tag**. Hai closing Tags ở đây là **</P>** và **</STRONG>**. Tất cả những gì nằm bên trong một cặp Tags được gọi là **Element**. Để nói thêm đặc tính của một Element, ta có thể nhét **Attribute** như **align** trong opening Tag của Element ấy dưới dạng **AttributeName="value"**, thí dụ như **align="center"**.

Vì Tags trong HTML được dùng để format (trình bày) tài liệu nên browser cần biết ý nghĩa của mỗi Tag. Một browser hay HTML parser sẽ thu thập các chỉ thị sau từ câu HTML trên:

1. Bắt đầu một Paragraph mới và đặt Text ở giữa trang (<P align="center">).
2. Hiển thị câu **Chào mừng bạn đến thăm**
3. Hiển thị chữ **Vovisoft** cách mạnh mẽ (<STRONG>Vovisoft</STRONG>).
4. Hiển thị câu **Web site**
5. Gặp điểm cuối của Paragraph (</P>)

Để xử lý đoạn code HTML trên, chẳng những browser cần phải xác định vị trí các Tags mà còn phải hiểu ý nghĩa của mỗi Tag. Vì mỗi Tag có ý ngĩa riêng của nó, thí dụ **P** cho Paragraph, **STRONG** để nhấn mạnh, thí dụ như dùng **chữ đậm (Bold)**.

Giống như HTML, XML đến từ SGML. Nó cũng dùng Tags để encode data. Điểm khác biệt chánh giữa HTML và XML là trong khi các Tags của HTML chứa ý nghĩa về **formatting (cách trình bày)** các dữ liệu, thì các Tags của XML chứa ý nghĩa về **cấu trúc** của các dữ liệu. Thí dụ như một tài liệu đặt hàng (order) XML dưới đây:

```
<Order OrderNo="1023">
   <OrderDate>2002-3-27</OrderDate>
   <Customer>Peter Collingwood</Customer>
   <Item>
      <ProductID>1</ProductID>
      <Quantity>5</Quantity>
   </Item>
   <Item>
      <ProductID>4</ProductID>
      <Quantity>3</Quantity>
   </Item>
</Order>
```

Tài liệu nầy chỉ chứa dữ liệu, không nhắc nhở gì đến cách trình bày. Điều nầy có nghĩa là một **XML parser** (chương trình ngắt khúc và phân tích) không cần phải hiểu ý nghĩa cũa các Tags. Nó chỉ cần tìm các Tags và xác định rằng đây là một tài liệu XML hợp lệ. Vì browser không cần phải hiểu ý nghĩa của các Tags, nên ta có thể dùng Tag nào cũng được. Đó là lý do người ta dùng chữ **eXtensible** (mở rộng thêm được), nhưng khi dùng chữ để viết tắt thì lại chọn **X** thay vì **e**, có lẽ vì X nghe có vẽ kỳ bí, hấp dẫn hơn.

Chúng ta hãy quan sát kỹ hơn cấu trúc của một XML. Trước hết, Element **Order** có Attribute **OrderNo** với value **1023**. Bên trong Element Order có:

- Một Child (con) Element **OrderDate** với value 2002-3-27
- Một Child Element **Customer** với value Peter Collingwood.
- Hai Child Elements **Item**, mỗi Element Item lại chứa một Child Element **ProductID** và một Child Element **Quantity**.

Đôi khi ta để một Element với tên đàng hoàng, nhưng không chứa một value, lý do là ta muốn dùng nó như một Element Nhiệm ý (Optional), có cũng được, không có cũng không sao. Cách tự nhiên nhất là gắn cái closing Tag ngay sau opening Tag. Thí dụ như **Empty (trống rỗng) Element** MiddleInitial trong Element customer dưới đây:

```
<Customer>
   <FirstName>Stephen</FirstName>
   <MiddleInitial></MiddleInitial>
   <LastName>King</LastName>
</Customer>
```

Có một cách khác để biểu diễn Empty Element là bỏ closing Tag và thêm một dấu "/" (slash) ở cuối openning Tag. Ta có thể viết lại thí dụ customer như sau:

```
<Customer>
   <FirstName>Stephen</FirstName>
   <MiddleInitial/>
   <LastName>King</LastName>
</Customer>
```

Dĩ nhiên Empty Element cũng có thể có Attribute như Element **PhoneNumber** thứ nhì dưới đây:

```
<Customer>
   <FirstName>Stephen</FirstName>
   <MiddleInitial></MiddleInitial>
   <LastName>King</LastName>
   <PhoneNumber Location="Home">9847 2635</PhoneNumber>
   <PhoneNumber Location="Work"></PhoneNumber>
</Customer>
```

Biểu diễn Data trong XML

Một tài liệu XML phải **well-formed** và **valid**. Mặc dầu hai từ nầy nghe tờ tợ, nhưng chúng có ý nghĩa khác nhau. Một XML well-formed là một XML thích hợp cho parser chế biến. Tức là XML tuân thủ các luật lệ về Tag, Element, Attribute , value .v.v.. chứa bên trong để parser có thể nhận diện và phân biệt mọi thứ.

Để ý là một XML well-formed chưa chắc chứa đựng những dữ liệu hữu dụng trong công việc làm ăn. Là well-formed chỉ có nghĩa là XML có cấu trúc đúng. Để hữu dụng cho công việc làm ăn, XML chẳng những well-formed mà còn cần phải valid. Một tài liệu XML valid khi nó chứa những data cần có trong loại tài liệu loại hay class ấy. Thí dụ một XML đặt hàng có thể bị đòi hỏi phải có một Attribute OrderNo và một Child Element Orderdate. Parser validate một XML bằng cách kiểm tra data trong XML xem có đúng như định nghĩa trong một Specification về loại tài liệu XML ấy. Specification nầy có thể là một **Document Type Definition (DTD)** hay một **Schema**.

Chốc nữa ta sẽ nói đến valid, bây giờ hãy bàn về well-formed.

Tạo một tài liệu XML well-formed

Để well-formed, một tài liệu XML phải theo đúng các luật sau đây:

1. Phải có một root (gốc) Element duy nhất, gọi là **Document Element**, nó chứa tất cả các Elements khác trong tài liệu.
2. Mỗi opening Tag phải có một closing Tag giống như nó.
3. Tags trong XML thì **case sensitive**, tức là opening Tag và closing Tag phải được đánh vần y như nhau, chữ hoa hay chữ thường.
4. Mỗi Child Element phải nằm trọn bên trong Element cha của nó.
5. Attribute value trong XML phải được gói giữa một cặp ngoặc kép hay một cặp apostrophe.

Luật thứ nhất đòi hỏi một root Element duy nhất, nên tài liệu dưới đây không well-formed vì nó không có một top level Element:

```
<Product ProductID="1">Chair</Product>
<Product ProductID="2">Desk</Product>
```

Một tài liệu XML không có root Element được gọi là một **XML fragment (mảnh)**. Để làm cho nó well-formed ta cần phải thêm một root Element như dưới đây:

```
<Catalog>
   <Product ProductID="1">Chair</Product>
   <Product ProductID="2">Desk</Product>
</Catalog>
```

Luật thứ hai nói rằng mỗi opening Tag phải có một closing Tag giống như nó. Tức là mỗi Tag mở ra phải được đóng lại. Empty Element viết cách gọn như <MiddleInitial/> được gọi là có Tag tự đóng lại. Các Tags khác phải có closing Tag. Cái XML dưới đây không well-formed vì nó có chứa một một Tag <Item> thiếu closing Tag </Item>:

```
<Order>
   <OrderDate>2002-6-14</OrderDate>
   <Customer>Helen Mooney</Customer>
   <Item>
      <ProductID>2</ProductID>
      <Quantity>1</Quantity>
   <Item>
      <ProductID>4</ProductID>
      <Quantity>3</Quantity>
   </Item>
</Order>
```

Để làm cho nó well-formed ta phải thêm cái closing tag cho Element Item thứ nhất:

```
<Order>
   <OrderDate>2002-6-14</OrderDate>
   <Customer>Helen Mooney</Customer>
   <Item>
      <ProductID>2</ProductID>
      <Quantity>1</Quantity>
   </Item>
   <Item>
      <ProductID>4</ProductID>
      <Quantity>3</Quantity>
   </Item>
</Order>
```

Luật thứ ba nói là tên Tag thì case sensitive, tức là closing Tag phải đánh vần y hệt như opening Tag, phân biệt chữ hoa, chữ thường. Như thế **<order>** khác với **<Order>**, ta không thể dùng Tag **</Order>** để đóng Tag **<order>**. Cái XML dưới đây không well-formed vì opening Tag và closing Tags của Element **OrderDate** không đánh vần giống nhau:

```
<Order>
   <OrderDate>2001-01-01</Orderdate>
   <Customer>Graeme Malcolm</Customer>
</Order>
```

Muốn làm cho nó well formed, ta phải sửa chữ **d** thành chữ hoa (uppercase) **D** như sau:

```
<Order>
   <OrderDate>2001-01-01</OrderDate>
   <Customer>Graeme Malcolm</Customer>
</Order>
```

Luật thứ tư nói mỗi Child Element phải nằm trọn bên trong Element cha của nó, tức là không thể bắt đầu một Element mới khi Element nầy chưa chấm dứt. Thí dụ như tài liệu XML dưới đây không well-formed vì closing Tag của **Category** hiện ra trước closing Tag của **Product**.

```
<Catalog>
```

```
   <Category CategoryName="Beverages">
       <Product ProductID="1">
          Coca-Cola
       </Category>
   </Product>
</Catalog>
```

Muốn sửa cho nó well-formed ta cần phải đóng Tag Product trước như dưới đây:

```
<Catalog>
   <Category CategoryName="Beverages">
       <Product ProductID="1">
          Coca-Cola
       </Product>
   </Category>
</Catalog>
```

Luật cuối cùng về tài liệu XML well-formed đòi hỏi value của Attribute phải được gói trong một cặp apostrophe hay ngoặc kép. Tài liệu dưới đây không well-form vì các Attribute values không được ngoặc đàng hoàng, số **1** không có dấu ngoặc, số **2** có một cái apostrophe, một cái ngoặc kép:

```
<Catalog>
   <Product ProductID=1>Chair</Product>
   <Product ProductID='2">Desk</Product>
</Catalog>
```

Processing Instructions và Comments

Ngoài các dữ liệu cần thiết cho công việc làm ăn, một tài liệu XML cũng có chứa các **Processing Instructions (chỉ thị về cách chế biến)** cho parser và **Comments (ghi chú)** cho người đọc.

Processing Instruction nằm trong cặp Tags **<?** và **?>**. Thông thường nó cho biết **version** của XML Specification mà parser cần làm theo. Có khi nó cũng cho biết data trong XML dùng **encoding** nào, thí dụ như uft-8. Còn một Attribute nữa là **standalone**. standalone cho parser biết là tài liệu XML có thể được validated một mình, không cần đến một DTD hay Schema.

Mặc dầu một tài liệu XML well-formed không cần có một Processing Instruction, nhưng thông thường ta để một Processing Instruction ở đàng đầu tài liệu, phần ấy được gọi là **prologue (giáo đầu)**. Dưới đây là một thí dụ có Processing Instruction trong prologue của một tài liệu XML:

```
<?xml version="1.0" encoding="utf-8"  standalone="yes"?>
<Order>
   <OrderDate>2002-6-14</OrderDate>
   <Customer>Helen Mooney</Customer>
   <Item>
      <ProductID>1</ProductID>
      <Quantity>2</Quantity>
   </Item>
   <Item>
      <ProductID>4</ProductID>
      <Quantity>1</Quantity>
   </Item>
</Order>
```

Có một loại Processing Instruction khác cũng rất thông dụng là cho biết tên của stylesheet của XML nầy, thí dụ như:

```
<?xml-stylesheet type="text/xsl" href="order.xsl"?>
```

Ở đây ta cho XML stylesheet parser biết rằng stylesheet thuộc loại **text/xsl** và nó được chứa trong file tên **order.xsl**. Bạn cũng có thể cho thêm **Comment** bằng cách dùng cặp Tags **<!--** và **-->** như sau:

```
<?xml version="1.0" encoding="utf-8"  standalone="yes"?>
```

```
<!-- Below are details of a purchase order.  -->
<Order>
   <OrderDate>2002-6-14</OrderDate>
   <Customer>Helen Mooney</Customer>
   <Item>
      <ProductID>1</ProductID>
      <Quantity>2</Quantity>
   </Item>
   <Item>
      <ProductID>4</ProductID>
      <Quantity>1</Quantity>
   </Item>
</Order>
```

Namespaces

Có một ý niệm rất quan trọng trong XML là **Namespace**. Nó cho ta cách cùng một tên của Element để nói đến hai thứ dữ liệu khác nhau trong cùng một tài liệu XML. Giống như có hai học sinh trùng tên **Tuấn** trong lớp học, ta phải dùng thêm họ của chúng để phân biệt, ta gọi Tuấn Trần hay Tuấn Lê. Thí dụ như có một order được người ta đặt trong tiệm sách như sau:

```
<?xml version="1.0"?>
<BookOrder OrderNo="1234">
   <OrderDate>2001-01-01</OrderDate>
   <Customer>
      <Title>Mr.</Title>
      <FirstName>Graeme</FirstName>
      <LastName>Malcolm</LastName>
   </Customer>
   <Book>
      <Title>Treasure Island</Title>
      <Author>Robert Louis Stevenson</Author>
   </Book>
</BookOrder>
```

Khi quan sát kỹ, ta thấy có thể có sự nhầm lẫn về cách dùng Element **Title**. Trong tài liệu có hai loại Title, một cái dùng cho khách hàng **Customer** nói đến danh hiệu **Mr., Mrs., Dr.**, còn cái kia để nói đến đề tựa của một quyển sách **Book**.

Để tránh sự lầm lẫn, bạn có thể dùng Namespace để nói rõ tên Element ấy thuộc về giòng họ nào. Giòng họ ấy là một **Universal Resource Identifier (URI)**. Một URI có thể là một URL hay một chỗ nào định nghĩa tính cách độc đáo của nó. Một namespace cũng không cần phải nói đến một địa chỉ Internet, nó chỉ cần phải là có một, không hai.

Bạn có thể khai báo namespaces trong một Element bằng cách dùng **Attribute xmlns** (**ns** trong chữ xmlns là viết tắt cho namespace) bạn cũng có thể khai báo một **default namespace** để áp dụng cho những gì nằm bên trong một Element, nơi bạn khai báo namespace. Thí dụ cái tài liệu đặt hàng có thể được viết lại như sau:

```
<?xml version="1.0"?>
<BookOrder OrderNo="1234">
   <OrderDate>2001-01-01</OrderDate>
   <Customer xmlns="http://www.northwindtraders.com/customer">
      <Title>Mr.</Title>
      <FirstName>Graeme</FirstName>
      <LastName>Malcolm</LastName>
   </Customer>
   <Book xmlns="http://www.northwindtraders.com/book">
      <Title>Treasure Island</Title>
      <Author>Robert Louis Stevenson</Author>
   </Book>
```

```
</BookOrder>
```

Ta đã tránh được sự nhầm lẫn vì bên trong Customer thì dùng namespace **http://www.northwindtraders.com/customer** và bên trong Book thì dùng namespace **http://www.northwindtraders.com/book**.

Tuy nhiên, ta sẽ giải quyết làm sao nếu trong order có nhiều customer và nhiều book. Nếu cứ thay đổi namespace hoài trong tài liệu thì chóng mặt chết. Một cách giải quyết là khai báo chữ viết tắt cho các namespaces ngay ở đầu tài liệu, trong root Element (tức là Document Element). Sau đó bên trong tài liệu ta sẽ prefix các Element cần xác nhận namespace bằng chữ viết tắt của namespace nó. Thí dụ như sau:

```
<?xml version="1.0"?>
<BookOrder  xmlns="http://www.northwindtraders.com/order"
       xmlns:cust="http://www.northwindtraders.com/customer"
       xmlns:book="http://www.northwindtraders.com/book" OrderNo="1234">
   <OrderDate>2001-01-01</OrderDate>
   <cust:Customer>
      <cust:Title>Mr.</cust:Title>
      <cust:FirstName>Graeme</cust:FirstName>
      <cust:LastName>Malcolm</cust:LastName>
   </cust:Customer>
   <book:Book>
      <book:Title>Treasure Island</book:Title>
      <book:Author>Robert Louis Stevenson</book:Author>
   </book:Book>
</BookOrder>
```

Trong tài liệu XML trên ta dùng 3 namespaces: một **default namespace** tên **http://www.northwindtraders.com/order**, namespace **http://www.northwindtraders.com/customer** (viết tắt là **cust**) và namespace **http://www.northwindtraders.com/book** (viết tắt là **book**). Các Elements và Attributes không có prefix (tức là không có chữ tắt đứng trước) như BookOrder, OrderNo, và OrderDate, được coi như thuộc về default namespace. Để đánh dấu một Element hay Attribute không thuộc về default namespace, một chữ tắt, đại diện namespace sẽ được gắn làm prefix cho tên Element hay Attribute. Thí dụ như **cust:LastName**, **book:Title**.

CDATA

CDATA là khúc dữ liệu trong tài liệu XML nằm giữa **<![CDATA[** và **]]>**. Data nằm bên trong những CDATA được cho thông qua parser y nguyên, không bị sửa đổi. Điểm nầy rất quan trọng khi bạn muốn cho vào những dữ liệu có chứa những text được xem như markup. Bạn có thể đặt những thí dụ cho XML trong những CDATA và chúng sẽ được parser bỏ qua. Khi dùng XSL stylesheets để transform một XML file thành HTML, có bất cứ scripting nào bạn cũng phải đặt trong những CDATA. Dưới đây là các thí dụ dùng CDATA:

```
<![CDATA[...place your data here...]]>

<SCRIPT>
  <![CDATA[
      function warning()
    {
        alert("Watch out!");
    }
   ]]>
</SCRIPT>
```

Entity References

Entity nói đến cách viết một số dấu đặc biệt đã được định nghĩa trước trong XML. Có 5 entities dưới đây:

Entity	Description
'	dấu apostrophe
&	dấu ampersand
>	dấu lớn hơn

<	dấu nhỏ hơn
"	dấu ngoặc kép

Trong bài tới ta sẽ học về cách process (chế biến) một tài liệu XML.

Đi lại trong XML bằng XPATH (phần I)

Chúng ta đã thấy cấu trúc và cú pháp của XML tương đối đơn giãn. XML cho ta một cách chuẩn để trao đổi tin tức giữa các computers. Bước tiếp theo là tìm hiểu cách nào một chương trình chế biến (process) một tài liệu XML

Dĩ nhiên để chế biến một XML chương trình ứng dụng phải có cách đi lại bên trong tài liệu để lấy ra values của các Elements hay Attributes. Do đó người ta thiết kế ra ngôn ngữ **XML Path language**, mà ta gọi tắt là **XPath**. XPath đóng một vai trò quan trọng trong công tác trao đổi dữ liệu giữa các computers hay giữa các chương trình ứng dụng vì nó cho phép ta lựa chọn hay sàng lọc ra những tin tức nào mình muốn để trao đổi hay hiển thị.

Nếu khi làm việc với cơ sở dữ liệu ta dùng SQL statement **Select .. from TableXYZ WHERE ...** để trích ra một số records từ một table, thì khi làm việc với XML, một table dữ liệu nho nhỏ, XPath cho ta những expressions về criteria (điều kiện) giống giống như clause **WHERE** trong SQL.

XPath là một chuẩn để process XML, cũng giống như SQL là một chuẩn để làm việc với cơ sở dữ liệu. Tiên phuông trong việc triển khai các chương trình áp dụng XPath là công tác của các công ty phần mềm lớn như Microsoft, Oracle, Sun, IBM, v.v. Sở dĩ ta cần có một chuẩn XPath là vì nó được áp dụng trong nhiều hoàn cảnh, nên cần phải có một lý thuyết rõ ràng, chính xác.

Lý thuyết về XPath hơi khô khan nhưng nó được áp dụng trong mọi kỹ thuật của gia đình XML. Cho nên bạn hãy kiên nhẫn nắm vững những điều căn bản về nó để khi nào gặp chỗ người ta dùng XPath thì mình nhận diện và hiểu được. So với võ thuật, thì XPath trong XML giống như Tấn pháp và cách thở. Tập luyện Tấn pháp thì mõi chân, tập thở thì nhàm chán, nhưng không có hai thứ đó thì ra chiêu không có công lực, chưa đánh đã thua rồi.

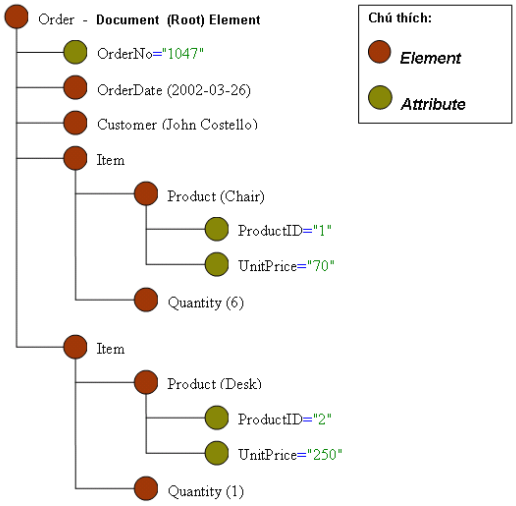
Ta sẽ chỉ học những thứ thường dùng trong XPath thôi, nếu bạn muốn có đầy đủ chi tiết về XPath thì có thể tham khảo Specification của nó ở http://www.w3c.org/TR/xpath.

XML như một cây đối với XPath

XPath cho ta cú pháp để diễn tả cách đi lại trong XML. Ta coi một tài liệu XML như được đại diện bằng một tree (cây) có nhiều nodes. Mỗi Element hay Attribute là một node. Để minh họa ý niệm nầy, bạn hãy quan sát tài liệu đặt hàng (order) XML sau:

```
<?xml version="1.0"?>
<Order OrderNo="1047">
   <OrderDate>2002-03-26</OrderDate>
   <Customer>John Costello</Customer>
   <Item>
      <Product ProductID="1" UnitPrice="70">Chair</Product>
      <Quantity>6</Quantity>
   </Item>
   <Item>
      <Product ProductID="2" UnitPrice="250">Desk</Product>
      <Quantity>1</Quantity>
   </Item>
</Order>
```

Ta có thể biểu diễn XML trên bằng một Tree như dưới đây, trong đó node Element màu nâu, node Attribute màu xanh:



Chỉ định Location Path

Bạn có thể dùng XPath expression để chỉ định **Location Path** (lối đi đến vị trí) đến node nào hay trích ra (trả về) một hay nhiều nodes thỏa đúng điều kiện yêu cầu. XPath expression có thể là **tuyệt đối**, tức là lấy node gốc làm chuẩn hay **tương đối**, tức là khởi đầu từ node vừa mới được chọn. Node ấy được gọi là **context node** (node vai chính trong tình huống).

Có hai cách viết để diễn tả XPath Location, viết nguyên và viết tắt. Trong cả hai cách ta đều dùng dấu slash (/) để nói đến **Document Element**, tức là node gốc. Ta có thể đi lại trong các node của Tree giống giống như các node của Windows System Directory mà ta thấy trong Panel bên trái của Window Explorer. Ta cũng sẽ dùng những ký hiệu như slash /, một chấm **.** và hai chấm **..** của Windows System File Folder cho cách viết tắt trong XPath Location để đi xuống các nodes con, cháu, chỉ định context node, hay đi ngược lên các nodes tổ tiên.

Location Path tuyệt đối

Chúng ta hãy tìm vài location paths trong cái Tree của tài liệu XML về đặt hàng nói trên. Muốn chọn cái node của Element *Order* (nó cũng là Root Element) bằng cú pháp nguyên, ta sẽ dùng XPath expression sau đây:

```
/child::Order
```

Dịch ra cú pháp tắt, expression nầy trở nên:

```
/Order
```

Đi ra nhánh của Tree, ta sẽ tìm được node *Customer* bằng cách dùng XPath expression sau:

```
/child::Order/child::Customer
```

Sau đây là XPath expression viết tắt tương đương:

```
/Order/Customer
```

Nếu bạn muốn lấy ra một node Attribute, bạn phải nói rõ điều nầy bằng cách dùng từ chìa khóa (keyword) **attribute** trong cách viết nguyên hay dùng character **@** trong cú pháp tắt. Do đó để lấy Attribute *OrderNo* của Element *Order*, ta sẽ dùng XPath expression sau:

```
/child::Order/attribute::OrderNo
```

Cú pháp tắt cho Attribute *OrderNo* là:

```
/Order/@OrderNo
```

Để trích ra các nodes con cháu, tức là các nodes nhánh xa hơn, ta dùng keyword **descendant** trong cú pháp nguyên hay một double slash (//) trong cú pháp tắt. Thí dụ, để lấy ra các nodes Product trong tài liệu, bạn có thể dùng expression location path sau:

```
/child::Order/descendant::Product
```

Cú pháp tắt tương đương là:

```
/Order//Product
```

Bạn cũng có thể dùng wildcards (lá bài Joker) để nói đến những nodes mà tên của chúng không thành vấn đề. Thí dụ, dấu asterisk (*) wildcard chỉ định bất cứ node tên nào. Location path sau đây chọn tất cả các nodes con của Element *Order*:

```
/child::Order/child::*
```

Cú pháp tắt tương đương là:

```
/Order/*
```

Location Path tương đối

Nhiều khi XPath location paths là tương đối với context node, trong trường hợp ấy location path diễn tả cách lấy ra một node hay một số (set of) nodes tương đối với context node. Thí dụ như, nếu Element *Item* thứ nhất trong *order* là context node, thì location path tương đối để trích ra Element con *Quantity* là:

```
child::Quantity
```

Trong cú pháp tắt, location path tương đối là:

```
Quantity
```

Tương tự như vậy, để lấy ra Attribute *ProductID* của Element con *Product*, cái location path tương đối là:

```
child::Product/attribute::ProductID
```

Expression ấy dịch ra cú pháp tắt là:

```
Product/@ProductID
```

Để đi ngược lên phía trên của Tree, ta dùng keyword **parent** (cha). Dạng tắt tương đương của keyword nầy là hai dấu chấm (..). Thí dụ nếu context node là Element *OrderDate*, thì Attribute *OrderNo* có thể được lấy ra từ Element *Order* bằng cách dùng location path tương đối sau:

```
parent::Order/attribute::OrderNo
```

Để ý là cú pháp nầy chỉ trả về một trị số khi node cha tên *Order*. Nếu muốn lấy ra Attribute *OrderNo* từ node cha không cần biết nó tên gì bạn phải dùng expression sau:

```
parent::*/attribute::OrderNo
```

Viết theo kiểu tắt đơn giản hơn vì bạn không cần phải cung cấp tên của node cha. Bạn có thể nói đến node cha bằng cách dùng hai dấu chấm (**..**) như sau:

```
../@OrderNo
```

Ngoài ra, bạn có thể nói đến chính context node bằng cách dùng hoặc keyword **self** hoặc một dấu chấm (**.**). Điều nầy rất tiện trong vài trường hợp, nhất là khi bạn muốn biết current context node là node nào.

Dùng điều kiện trong Location Path

Bạn có thể giới hạn số nodes lấy về bằng cách gắn thêm điều kiện sàng lọc vào location path. Cái điều kiện giới hạn một hay nhiều nodes được tháp vào expression bên trong một cặp ngoặc vuông (**[]**). Thí dụ, để lấy ra mọi Element *Product* có Attribute *UnitPrice* lớn hơn 70, bạn có thể dùng XPath expression sau đây:

```
/child::Order/child::Item/child::Product[attribute::UnitPrice>70]
```

Trong cú pháp tắt, nó là:

```
/Order/Item/Product[@UnitPrice>70]
```

Trong expression của điều kiện bạn cũng có thể dùng Xpath tương đối , do đó trong expression điều kiện bạn có thể dùng bất cứ node nào trong thứ bậc. Thí dụ sau đây lấy về những nodes *Item* có Element con *Product* với Attibute *ProductID* trị số bằng 1:

```
/child::Order/child::Item[child::Product/attribute::ProductID=1]
```

Dịch ra cú pháp tắt, ta có:

```
/Order/Item[Product/@ProductID=1]
```

Đi lại trong XML bằng XPATH (phần II)

Collections

Cái bộ (Set of) Nodes do XPath trả về được gọi là **Collection**. Thông thường trong lập trình, từ "Collection" được dùng để nói đến một tập hợp các objects đồng loại. Ta có thể lần lượt đi qua (iterate through) các objects trong một Collection nhưng không được bảo đảm thứ tự của chúng, tức là gặp object nào trước hay object nào sau.

Trái lại, trong chuẩn XPath, khi một Collection được trả về bởi một XPath Query (hỏi), nó giữ nguyên thứ tự các Nodes và cấp bậc của chúng trong tài liệu XML. Tức là nếu XPath trả về một cành các nodes thì trừ những nodes không thỏa điều kiện, các node còn lại vẫn giữ đúng vị trí trên cành.

Vì các Attributes của một Element không có thứ tự, nên chúng có thể nằm lộn xộn trong một Collection.

Indexing trong một Collection

Một Collection của Nodes được xem như một Array. Muốn nói trực tiếp đến một Node trong Collection ta có thể dùng một index trong cặp ngoặc vuông. Node thứ nhất có Index là 1.

Cặp ngoặc vuông ([]) có precedence cao hơn (được tính trước) dấu slash(/) hay hai dấu slash (//). Dưới đây là hai thí dụ:

Expression	Ý nghĩa
author[1]	Element **author** đầu tiên.
author[firstname][3]	Element **author** thứ ba có một Element **firstname** con.

Mối liên hệ (Axes)

Một location path dùng một **Axis** để chỉ định mối liên hệ giữa các Nodes được chọn đối với context node. Sau đây là bảng liệt kê đầy đủ các axes:

Axes	Ý nghĩa
ancestor::	Tổ tiên của context node. Những tổ tiên của context node gồm có cha, ông nội, ông cố .v.v., do đó ancestor:: axis luôn luôn kể cả root node trừ khi chính context node là root node.
ancestor-or-self::	Chính context node và tổ tiên của nó. Cái ancestor-or-self:: axis luôn luôn kể cả root node.
attribute::	Các Attributes của context node. Nếu context node không phải là một Element thì chắc chắn axis sẽ trống rỗng.
child::	Con cái của context node. Một con là bất cứ node nào nằm ngay dưới context node trong tree. Tuy nhiên, Attribute hay Namespace nodes không được xem là con cái của context node.
descendant::	Con cháu của context node. Con cháu là con, cháu, chít, .v.v., do đó descendant:: axis không bao giờ chứa Attribute hay Namespace nodes.
following::	Mọi nodes hiện ra sau context node trên tree, không kể con cháu, Attribute nodes, hay Namespace nodes.
following-sibling::	Mọi nodes em (nằm sau) context node. following-sibling:: axis nói đến chỉ những Nodes con, của cùng một Node cha, nằm trên tree sau context node. Axis không kể các Nodes anh nằm trước context node. Nếu context node là Attribute hay Namespace thì following-sibling:: axis sẽ trống rỗng.
namespace::	Những Namespace nodes của context node. Mỗi namespace có một namespace node trong scope (phạm vi hoạt động) của context node. Nếu context node không phải là một Element thì Axis sẽ trống rỗng.
parent::	Node cha của context node, nếu nó có cha. Node cha là node nằm ngay phía trên context node trên tree.
preceding::	Mọi nodes hiện ra trước context node trên tree, không kể các nodes tổ tiên, Attribute nodes, hay Namespace nodes.

	Một cách để nhận diện preceding:: axis là mọi nodes đã kết thúc hoàn toàn trước khi context node bắt đầu.
preceding-sibling::	Mọi nodes anh (nằm trước) context node. preceding-sibling:: axis nói đến chỉ những Nodes con, của cùng một Node cha, nằm trên tree trước context node. Nếu context node là Attribute hay Namespace thì preceding-sibling:: axis sẽ trống rỗng.
self::	Là chính context node.

Sàng lọc (Filters)

Như ta đã thấy ở trên, để giới hạn chỉ lấy ra những Nodes thỏa đáng một điều kiện, ta gắn một **Filter** (sàng lọc) vào Collection. Filter ấy là một Clause giống giống **Clause WHERE** trong ngôn ngữ SQL của cơ sở dữ liệu.

Nếu một Collection nằm giữa một filter, nó sẽ cho kết quả TRUE nếu Collection trả về ít nhất một Node và FALSE nếu Collection trống rỗng (empty). Thí dụ expression **author/degree** có nghĩa rằng hàm ***biến đổi Collection ra trị số Boolean*** sẽ có giá trị TRUE nếu hiện hữa một Element **author** có Element con tên **degree**.

Filters luôn luôn được tính theo context của nó. Nói một cách khác, cái expression **book[author]** có nghĩa là cho mỗi Element **book** tìm thấy, nó sẽ được thử xem có chứa một Element con tên **author** không. Tương tự như vậy, **book[author = 'Brown']** có nghĩa rằng cho mỗi Element **book** tìm thấy, nó sẽ được thử xem có chứa một Element con tên **author** với trị số bằng **Brown** không.

Ta có thể dùng dấu chấm (.) để khám current context node. Thí dụ như, **book[. = 'Dreams']** có nghĩa rằng cho mỗi Element book tìm thấy trong current context, nó sẽ được thử xem có trị số bằng **Dreams** không. Dưới đây là một ít thí dụ:

Expression	Ý nghĩa
book[excerpt]	Mọi Element **book** có chứa ít nhất một Element **excerpt**.
book[excerpt]/title	Mọi Element **title** nằm trong những Element **book** có chứa ít nhất một Element **excerpt**.
book[excerpt]/author[degree]	Mọi Element **author** có chứa ít nhất một Element **degree** và nằm trong những Elements **book** có chứa ít nhất một Element **excerpt**.
book[author/degree]	Mọi Element **book** có chứa ít nhất một Element **author** với ít nhất một Element **degree** con.
book[excerpt][title]	Mọi Element **book** có chứa ít nhất một Element **excerpt** và ít nhất một Element **title**.

So sánh

Để so sánh hai objects trong XPath ta dùng dấu (=) cho **bằng nhau** và (**!=**) cho **không bằng nhau**. Mọi Element và Attributes là **string**, nhưng được Typecast (xem như) những con số khi đem ra so sánh.

Expression	Ý nghĩa
author[lastname = "Smith"]	Mọi Element **author** có chứa ít nhất một Element **lastname** với trị số bằng **Smith**.
author[lastname[1] = "Smith"]	Mọi Element **author** có Element **lastname** con đầu tiên với trị số bằng **Smith**.
author/degree[@from != "Harvard"]	Mọi Element **degree**, là con một Element **author**, và có một Attribute **from** với trị số không phải là **"Harvard"**.

author[lastname = /editor/lastname]	Mọi Element **author** có chứa một Element **lastname** bằng với Element **lastname** là con của root Element **editor**.
author[. = "John Hamilton"]	Mọi Element **author** có trị số string là **John Hamilton**.

Operator Union | (họp lại)

Ngôn ngữ Xpath hỗ trợ Operator Union, giống như Logical **OR (hoặc là)**. Dưới đây là vài thí dụ:

Expression	Ý nghĩa
firstname | lastname	Mọi Element **firstname** và **lastname** trong current context.
(bookstore/book | bookstore/magazine)	Mọi Element **book** hay **magazine** là con một Element **bookstore**.
book | book/author	Mọi Element **book** hay Element **author** là con những Elements **book**.
(book | magazine)/price	Mọi Element **price** là con của Element **book** hay Element **magazine**.

Thử loại Node (Node Type Tests)

Để chọn những loại Node khác hơn là Element node, ta dùng **Node-Type Test**. Mục đích của việc dùng Node-Type test là để chỉ định sự lựa chọn khác thường. Thí dụ như, **descendant::text()** cho ta mọi text nodes là con cháu của context node, dù rằng loại node chính của con cháu context node là Element. Có 4 loại Node-Type tests như liệt kê dưới đây.

Node type	Trả về	Thí dụ
comment()	mọi comment node.	**following::comment()** chọn mọi comment nodes hiện ra sau context node.
node()	mọi node.	**preceding::node()** chọn mọi nodes hiện ra trước context node.
processing-instruction()	mọi processing instruction node.	**self::processing instruction()** chọn mọi processing instruction nodes trong context node.
text()	mọi text node.	**child::text()** chọn mọi text nodes là con của the context node.

Thử Node nhắm vào loại Processing Instruction

Một node test có thể chọn processing instruction thuộc loại nào, tức là chọn **mục tiêu (target)**. Cú pháp của một loại test như thế là:

```
processing-instruction("target")
```

Thí dụ node test sau đây trả về mọi processing instruction nodes có nhắc đến một XSL stylesheet trong tài liệu:

```
/child::processing-instruction("xml-stylesheet")
```

Thêm một số thí dụ Location Path

Expression	Ý nghĩa
./author	Mọi Element **author** trong current context. Expresion nầy tương đương với expression trong hàng kế.
author	Mọi Element **author** trong current context.

/bookstore	Document (Root) Element tên **bookstore** của tài liệu nầy.
//author	Mọi Element **author** trong tài liệu.
book[/bookstore/@specialty = @style]	Mọi Element **book** có Attribute **style** với value bằng value của Attribute **specialty** của Document Element **bookstore** của tài liệu.
author/firstname	Mọi Element **firstname** con của các Elements **author**.
bookstore//title	Mọi Element **title** một hay nhiều bậc thấp hơn, tức là con cháu của, Element **bookstore**. Lưu ý là expression nầy khác với expression trong hàng kế.
bookstore/*/title	Mọi Element **title** cháu của các **bookstore**.
bookstore//book/excerpt//emph	Mọi Element **emph** bất cứ nơi nào dưới **excerpt** là con của những elements **book** , bất cứ nơi nào dưới element **bookstore**.
.//title	Mọi Element **title** một hay nhiều bậc thấp hơn current context node.
author/*	Mọi Element là con của các elements con **author**.
book/*/lastname	Mọi Element **lastname** là cháu của các elements con **book**.
/	Mọi Element cháu của current context node.
***[@specialty]**	Mọi Element con có Attribute **specialty**.
@style	Attribute **style** của current context node.
price/@exchange	Attribute **exchange** của những Elements **price** trong current context, tức là những Elements **price** của current context node.
price/@exchange/total	Trả về một node set trống rỗng, vì Attributes không có Element con. Expression nầy được chấp nhận trong văn phạm của XML Path Language, nhưng không thật sự hợp lệ.
book[@style]	Mọi Element **book** có Attribute **style** trong current context node. Lưu ý phần nằm trong ngoặc vuông là điều kiện của Element book
book/@style	Attribute **style** của mọi Element **book**trong current context node. Ở đây không có điều kiện như hàng trên. Ta nói đến Attribute hay Element nằm bên phải nhất.
@*	Mọi Attributes của current context node.
author[1]	Element **author** thứ nhất trong current context node.
author[firstname][3]	Element **author** thứ ba có một Element con **firstname**.
my:book	Element **book** từ namespace **my**.
my:*	Mọi Element trong namespace **my**.

XSL Style Sheets (phần I)

X ML là cách tuyệt diệu cho ta sắp xếp dữ liệu để trao đổi chúng giữa các tổ chức và giữa các chương trình ứng dụng. Tuy nhiên, chẳng chóng thì chầy, ta sẽ khám phá sự đa diện của cơ sở dữ liệu khắp nơi. Và ngay cả có chuẩn XML rồi, ta vẫn cần một công cụ hiệu lực để trình bày dữ liệu trong nhiều kiểu khác nhau thích hợp cho áp dụng chế biến ở một nơi khác.

XSL - eXtensible Style Sheet (những trang diễn tả dáng điệu) là một ngôn ngữ chuẩn giúp ta **biến đổi (transform)** một tài liệu XML ra format khác, như HTML, Wireless (vô tuyến điện) Markup Language (WML),

và ngay cả một XML khác. Lúc nguyên thủy, XSL được thiết kế để sanh ra nhiều HTML trong những dạng khác nhau tùy theo Style sheet. Tức là XSL thêm dáng điệu cho XML, vì chính bản chất của XML chỉ là một cấu trúc của những mảnh dữ liệu.

Thí dụ ta có hai Style sheet versions cho một XML, một cái dùng để tạo ra HTML cho trang Web thông thường trên computer, còn cái kia để tạo ra trang Web dùng cho Mobile Phone hay Pocket PC, những dụng cụ có màn ảnh nhỏ. Cả hai trang Web đều chứa cùng một số dữ liệu, có thể trên màn ảnh nhỏ thì giới hạn những dữ liệu quan trọng thôi, nhưng cách trình bày có thể rất khác nhau.

Tuy nhiên, sau đó không lâu, người ta thấy XML có thể được XSL biến đổi ra bất cứ Output Format nào, ngay cả chính XML. Có một version mới, rất hay của XSL vừa ra đời. Nó được gọi là **XSL Transformations (XSLT)**.

Chúng ta sẽ lần lượt học các cú pháp thông dụng của XSL. Tuy không nhiều, nhưng nó giúp bạn có một ý niệm căn bản về kỹ thuật nầy để bạn có thể bắt đầu dùng XSL style sheets biến chế dữ liệu trong tài liệu XML. Muốn có một XSL reference đầy đủ , bạn có thể thăm trang http://www.w3.org/Style/XSL.

Nên nhớ là giống như XPath, XSL và XSLT chỉ là những tiêu chuẩn ấn định những gì ta đòi hỏi một chương trình áp dụng được thực hiện để hổ trợ chúng cần phải có. Tuy nhiên, ai triển khai chương trình đó, và bằng ngôn ngữ lập trình nào cũng được. Thí dụ như Microsoft cho ta MSXML version 3 để dùng XSL và XSLT.

Những trang XSL Style Sheet

Những trang XSL định nghĩa những **style sheets (trang dáng điệu)** để ta có thể áp dụng vào những tài liệu XML. Một style sheet chứa những chỉ dẫn (instructions) để bảo một XML parser làm cách nào phát sinh (generate) ra một tài liệu trình duyệt kết quả cho những dữ liệu trong một tài liệu XML.

Bản thân XSL style sheet cũng là một XML well-formed nhưng nó chứa những lệnh (commands) XSL và những câu HTML text dùng y nguyên cho output.

Để XML parser nhận diện được các lệnh trong một XSL, bạn phải khai báo (declare) một **namespace** trong root element, thường thường với một prefix **xsl**. Một Style sheet thường thường chứa một trong hai namespaces: cái namespace **XSL** nguyên thủy (**http://www.w3.org/TR/WD-xsl**) hay cái namespace mới **XSLT** (**http://www.w3.org/1999/XSL/Transform**). Microsoft XML parser (**MSXML**) từ version 3.0 trở lên đều hỗ trợ cả hai namespaces.

Xin lưu ý là **Internet Explorer version 5.x** dùng MSXML 2.5, nên không hỗ trợ namespace XSLT. Muốn khắc phục trở ngại ấy, hoặc là bạn cài đặt Internet Explorer version 6, hoặc là bạn cài MSXML3 trong **Replace mode** bằng cách dùng công cụ tên **Xmlinst.exe** để thêm chức năng hỗ trợ namespace XSLT trong IE v5.x.

Cái **Root Element** trong một tài liệu XSL document thường thường là một **Element stylesheet**. Nó chứa một hay nhiều **Element Template** để được **matched** (cặp đôi vì giống nhau) với dữ liệu trong tài liệu XML, thí dụ như tài liệu đặt hàng (order) dưới đây:

```
<?xml version="1.0"?>
<Order OrderNo="1047">
   <OrderDate>2002-03-26</OrderDate>
   <Customer>John Costello</Customer>
   <Item>
      <Product ProductID="1" UnitPrice="70">Chair</Product>
      <Quantity>6</Quantity>
   </Item>
   <Item>
      <Product ProductID="2" UnitPrice="250">Desk</Product>
      <Quantity>1</Quantity>
   </Item>
</Order>
```

Vì chính XSL style sheet cũng là một tài liệu XML, nên nó phải tuân theo mọi luật về một XML well-formed. Sau đây là một XSL style sheet đơn giản có thể được áp dụng vào tài liệu order:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:template match="/">
      <HTML>
         <HEAD>
            <TITLE>Northwind Home Page</TITLE>
         </HEAD>
         <BODY>
            <P>Customer Order</P>
         </BODY>
      </HTML>
   </xsl:template>
</xsl:stylesheet>
```

Style sheet nầy dựa trên namespace XSLT và chứa vỏn vẹn một template (bảng kẻm in) được áp dụng vào Root (biểu hiệu bằng dấu slash / là trị số của Attribute **match**) của tài liệu XML và mọi Element bên trong của nó.

Một template thật thì gồm có một loạt Tags HTML sẽ hiện ra trong hồ sơ kết quả, nhưng trong trường hợp nầy cái Template không làm chuyện gì hữu ích; nó chỉ output (cho ra) một tài liệu HTML y nguyên như nằm trong XSL và không có chứa dữ liệu gì từ hồ sơ input XML. Để merge (hòa đồng) các dữ liệu trong XML vào XSL template, bạn cần phải dùng một ít lệnh (commands) XSL.

Lệnh *value-of*

XSL định nghĩa một số lệnh chế biến (processing commands) để trích dữ liệu ra từ một tài liệu XML và hòa nó vào một hồ sơ kết quả. Cái lệnh căn bản và hữu dụng nhất trong số nầy là lệnh **value-of**. Lệnh **value-of** chọn trị số (value) của một Element hay Attribute nào đó trong XML và hòa nó với hồ sơ output.

Lệnh **value-of** có dạng một XML Element trong XSL. Nó dùng một Attribute tên **select** có value là một XPath Location Path để trích ra một Node. Kết quả là **value của (value-of) Node** ấy. Do đó, khá hơn lần trước, bây giờ ta có thể trình bày dữ liệu của XML với lệnh value-of như sau:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:template match="/">
      <HTML>
         <HEAD>
            <TITLE>Northwind Home Page</TITLE>
         </HEAD>
         <BODY>
            <P>Customer Order</P>
            <P>Order No:
               <xsl:value-of select="Order/@OrderNo"/>
            </P>
            <P>Date:
               <xsl:value-of select="Order/OrderDate"/>
            </P>
            <P>Customer:
               <xsl:value-of select="Order/Customer"/>
            </P>
         </BODY>
      </HTML>
   </xsl:template>
</xsl:stylesheet>
```

Cái Style sheet kỳ nầy trích ra Attribute OrderNo và trị số của các Elements OrderDate và Customer từ Element Order bằng cách dùng một XPath location path. Lưu ý là các XPath expressions ở đây thì tương đối với context node chỉ định trong match parameter của **Element template** (trong trường hợp nầy là Root Element, biểu hiệu

bằng dấu slash /).

Áp dụng Style sheet nầy vào hồ sơ đặt hàng (order) XML ta sẽ được HTML sau đây:

```
<HTML>
   <HEAD>
      <TITLE>Northwind Home Page</TITLE>
   </HEAD>
   <BODY>
      <P>Customer Order</P>
      <P>Order No: 1047</P>
      <P>Date: 2002-03-26</P>
      <P>Customer: John Costello</P>
   </BODY>
</HTML>
```

Lệnh *for-each*

Trong một tài liệu XML, có thể có nhiều Elements mang cùng một tên để nói đến một danh sách những thứ tương tư. Thí dụ trong tài liệu đặt hàng có hai Element Item để diễn tả hai món hàng được đặt.

Hầu hết ngôn ngữ lập trình cho ta phương tiện để áp dụng cùng một cách chế biến cho mọi món trong nhóm. Như trong Visual Basic ta có FOR loop hay DO loop để iterate qua từng món trong bộ. Trong XSL cũng thế, bạn có thể dùng lệnh **for-each** để đi lần lượt qua từng Element trong nhóm, bằng cách dùng **Attribute select** để chỉ định những nodes mà bạn muốn làm việc.

Thí dụ ta có thể làm cho cái Style sheet hay hơn bằng cách liệt kê các Item trong Order thành một table:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:template match="/">
      <HTML>
         <HEAD>
            <TITLE>Northwind Home Page</TITLE>
         </HEAD>
         <BODY>
            <P>Customer Order</P>
            <P>Order No:
               <xsl:value-of select="Order/@OrderNo"/>
            </P>
            <P>Date:
               <xsl:value-of select="Order/OrderDate"/>
            </P>
            <P>Customer:
               <xsl:value-of select="Order/Customer"/>
            </P>
            <TABLE Border="0">
               <TR>
                  <TD>ProductID</TD>
                  <TD>Product Name</TD>
                  <TD>Price</TD>
                  <TD>Quantity Ordered</TD>
               </TR>
               <xsl:for-each select="Order/Item">
                  <TR>
                     <TD>
                        <xsl:value-of select="Product/@ProductID"/>
                     </TD>
                     <TD>
                        <xsl:value-of select="Product"/>
```

```
                    </TD>
                    <TD>
                       <xsl:value-of select="Product/@UnitPrice"/>
                    </TD>
                    <TD>
                       <xsl:value-of select="Quantity"/>
                    </TD>
                 </TR>
              </xsl:for-each>
           </TABLE>
        </BODY>
     </HTML>
   </xsl:template>
</xsl:stylesheet>
```

Lần nầy trong Style sheet, ta bảo parser đi qua từng Element Item để lấy ra Attributes ProductID và UnitPrice của Element Product , và values của Elements Product và Quantity, rồi cho vào table.

Lưu ý ở đây các XPath expressions tương đối dùng cái Node chỉ định trong lệnh **for-each** làm context node. Trong trường hợp nầy nó là Node Item. Cuối của **for-each** loop là closing Tag của Element for-each (</xsl:for-each>) . Style sheet trên nầy khi áp dụng vào tài liệu đặt hàng sẽ cho ra HTML sau đây:

```
<HTML>
   <HEAD>
      <TITLE>Northwind Home Page</TITLE>
   </HEAD>
   <BODY>
      <P>Customer Order</P>
      <P>Order No: 1047</P>
      <P>Date: 2002-03-26</P>
      <P>Customer: John Costello</P>
      <TABLE Border="0">
         <TR>
            <TD>ProductID</TD>
            <TD>Product Name</TD>
            <TD>Price</TD>
            <TD>Quantity Ordered</TD>
         </TR>
         <TR>
            <TD>1</TD>
            <TD>Chair</TD>
            <TD>70</TD>
            <TD>6</TD>
         </TR>
         <TR>
            <TD>2</TD>
            <TD>Desk</TD>
            <TD>250</TD>
            <TD>1</TD>
         </TR>
      </TABLE>
   </BODY>
</HTML>
```

Phần **BODY** của HTML trên hiển thị như sau:

Customer Order

Order No: 1047

Date: 2002-03-26

Customer: John Costello

ProductID	Product Name	Price	Quantity Ordered
1	Chair	70	6
2	Desk	250	1

Lệnh *Attribute*

Đôi khi ta muốn tạo ra thêm một Attribute trong hồ sơ output với một trị số lấy từ tài liệu XML input. Thí dụ như tương ứng với mỗi tên của một Product, bạn muốn tạo ra một **hyperlink** để chuyển (pass) cái **ProductID** qua một trang Web khác, nơi đó sẽ hiển thị chi tiết về mặt hàng nầy.

Để tạo ra một hyperlink trong một hồ sơ HTML, bạn cần tạo ra một **Element A (Anchor)** với một **Attribute href**. Bạn có thể dùng **lệnh Attribute** của XSL để thực hiện chuyện ấy như minh họa trong Style sheet dưới đây:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:template match="/">
      <HTML>
         <HEAD>
            <TITLE>Northwind Home Page</TITLE>
         </HEAD>
         <BODY>
            <P>Customer Order</P>
            <P>Order No:
               <xsl:value-of select="Order/@OrderNo"/>
            </P>
            <P>Date:
               <xsl:value-of select="Order/OrderDate"/>
            </P>
            <P>Customer:
               <xsl:value-of select="Order/Customer"/>
            </P>
            <TABLE Border="0">
               <TR>
                  <TD>ProductID</TD>
                  <TD>Product Name</TD>
                  <TD>Price</TD>
                  <TD>Quantity Ordered</TD>
               </TR>
               <xsl:for-each select="Order/Item">
                  <TR>
                     <TD>
                        <xsl:value-of select="Product/@ProductID"/>
                     </TD>
                     <TD>
                        <A>
                           <xsl:attribute name="HREF">Products.asp?ProductID=
                              <xsl:value-of select="Product/@ProductID"/>
                           </xsl:attribute>
                           <xsl:value-of select="Product"/>
                        </A>
                     </TD>
```

```
                        <TD>
                           <xsl:value-of select="Product/@UnitPrice"/>
                        </TD>
                        <TD>
                           <xsl:value-of select="Quantity"/>
                        </TD>
                     </TR>
                  </xsl:for-each>
               </TABLE>
            </BODY>
         </HTML>
      </xsl:template>
</xsl:stylesheet>
```

Áp dụng Style sheet nầy vào tài liệu đặt hàng XML, bạn sẽ có hồ sơ HTML sau:

```
<HTML>
   <HEAD>
      <TITLE>Northwind Home Page</TITLE>
   </HEAD>
   <BODY>
      <P>Customer Order</P>
      <P>Order No: 1047</P>
      <P>Date: 2002-03-26</P>
      <P>Customer: John Costello</P>
      <TABLE Border="0">
         <TR>
            <TD>ProductID</TD>
            <TD>Product Name</TD>
            <TD>Price</TD>
            <TD>Quantity Ordered</TD>
         </TR>
         <TR>
            <TD>1</TD>
            <TD>
               <A HREF="Products.asp?ProductID=1">Chair</A>
            </TD>
            <TD>70</TD>
            <TD>6</TD>
         </TR>
         <TR>
            <TD>2</TD>
            <TD>
               <A HREF="Products.asp?ProductID=2">Desk</A>
            </TD>
            <TD>250</TD>
            <TD>1</TD>
         </TR>
      </TABLE>
   </BODY>
</HTML>
```

Phần **BODY** của HTML trên hiển thị như sau:

Customer Order

Order No: 1047

Date: 2002-03-26

Customer: John Costello

ProductID	Product Name	Price	Quantity Ordered
1	Chair	70	6
2	Desk	250	1

Bạn có thể để Mouse cursor lên chữ **Chair** hay chữ **Desk** để thấy tên hyperlink của chúng hiển thị trong status bar của browser.

XSL Style Sheets (phần II)

Các lệnh về điều kiện

Giống như trong ngôn ngữ lập trình thông thường ta có các instructions về điều kiện như IF, SELECT CASE, ELSE .v.v.. để lựa chọn, trong XSL ta có các lệnh về điều kiện như ***xsl:if, xsl:choose, xsl:when***, và ***xsl:otherwise***. Khi expression của Element *xsl:if*, *xsl:when*, hay *xsl:otherwise* có trị số **true**, thì cái Template nằm bên trong nó sẽ được tạo ra (instantiated).

Thường thường, nếu công việc thử tính đơn giản ta dùng *xsl:if*. Nếu nó hơi rắc rối vì tùy theo trường hợp ta phải làm những công tác khác nhau thì ta dùng *choose/when/otherwise*.

Trị số của Attribute **test** của *xsl:if* và *xsl:when* là một expression để tính. Expression nầy có thể là một so sánh hay một expression loại **XPath**. Kết quả việc tính nầy sẽ là **true** nếu nó trả về một trong các trị số sau đây:

- Một bộ node có ít nhất một node
- Một con số khác zero
- Một mảnh (fragment) Tree
- Một text string không phải là trống rỗng (non-empty)

Để minh họa cách dùng các lệnh XSL về điều kiện ta sẽ dùng hồ sơ nguồn tên **catalog.xml** sau đây:

```
<?xml version="1.0"?>
<catalog>
   <book id="bk102">
      <author>Ralls, Kim</author>
      <title>Midnight Rain</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2000-12-16</publish_date>
      <description>A former architect battles corporate zombies, an evil sorceress, and
her own
childhood to become queen of the world.</description>
   </book>
   <book id="bk107">
      <author>Thurman, Paula</author>
      <title>Splish Splash</title>
      <genre>Romance</genre>
      <price>4.95</price>
      <publish_date>2000-11-02</publish_date>
      <description>A deep sea diver finds true love twenty thousand leagues beneath the
sea.</description>
   </book>
```

```
   <book id="bk108">
      <author>Knorr, Stefan</author>
      <title>Creepy Crawlies</title>
      <genre>Horror</genre>
      <price>4.95</price>
      <publish_date>2000-12-06</publish_date>
      <description>An anthology of horror stories about roaches, centipedes, scorpions
and other
insects.</description>
   </book>
   <book id="bk109">
      <author>Kress, Peter</author>
      <title>Paradox Lost</title>
      <genre>Science Fiction</genre>
      <price>6.95</price>
      <publish_date>2000-11-02</publish_date>
      <description>After an inadvertant trip through a Heisenberg Uncertainty Device,
James Salway
discovers the problems of being quantum.</description>
   </book>
   <book id="bk110">
      <author>O'Brien, Tim</author>
      <title>Microsoft .NET: The Programming Bible</title>
      <genre>Computer</genre>
      <price>36.95</price>
      <publish_date>2000-12-09</publish_date>
      <description>Microsoft's .NET initiative is explored in detail in this deep
programmer's
reference.</description>
   </book>
</catalog>
```

Dưới đây là một thí dụ dùng *xsl:if*:

```
<xsl:for-each select="//book">
   <tr>
      <td>
         <xsl:value-of select="title"/>
      </td>
      <td>
         <xsl:if test="price > 6">
            <xsl:attribute name="bgcolor">cyan</xsl:attribute>
         </xsl:if>
         <xsl:value-of select="price"/>
      </td>
   </tr>
</xsl:for-each>
```

Trong thí dụ trên, Attribute **bgcolor** chỉ được tạo ra với trị số **cyan** khi price của book lớn hơn 6. Mục đích của ta là dùng màu xanh da trời nhạt để làm nền cho sách nào có giá (**price**) cao hơn 6.

Dưới đây là một thí dụ dùng *xsl:choose*:

```
<xsl:for-each select="//book">
   <div>
      <xsl:choose>
         <xsl:when test="self::*[genre = 'Romance']">
            <xsl:attribute name="style">background-color: pink</xsl:attribute>
         </xsl:when>
         <xsl:when test="self::*[genre = 'Fantasy']">
```

```
            <xsl:attribute name="style">background-color: lightblue</xsl:attribute>
         </xsl:when>
         <xsl:otherwise>
            <xsl:attribute name="style">background-color: lightgreen</xsl:attribute>
         </xsl:otherwise>
      </xsl:choose>
      <xsl:value-of select="title"/>
   </div>
</xsl:for-each>
```

Trong thí dụ trên Attribute **style** của Cascading Style Sheet sẽ có những trị số cho background-color khác nhau tùy theo loại sách. Nếu là Romance thì pink, Fantasy thì lightblue, còn nếu không phải là Romance hay Fantasy (tức là *xsl:otherwise*) thì lightgreen. Màu nầy sẽ được dùng làm nền cho đề mục (**title**) của sách. Để ý là cặp Tags **<xsl:choose>,</xsl:choose>** được dùng để gói các *xsl:when*, và *xsl:otherwise* bên trong.

Sau đây là listing của một **catalog.xsl** style sheet đầy đủ, trong đó có cả hai cách dùng *xsl:if* và *xsl:when* nói trên:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:template match="/">
      <HTML>
         <HEAD>
            <TITLE>Book Lovers' Catalog</TITLE>
         </HEAD>
         <BODY>
            <Center>
               <H1>Book Lovers' Catalog</H1>
            </Center>
            <TABLE Border="1" Cellpadding="5">
               <TR>
                  <TD align="center" bgcolor="silver">
                     <b>ID</b>
                  </TD>
                  <TD align="center" bgcolor="silver">
                     <b>Author</b>
                  </TD>
                  <TD align="center" bgcolor="silver">
                     <b>Title</b>
                  </TD>
                  <TD align="center" bgcolor="silver">
                     <b>Genre</b>
                  </TD>
                  <TD align="center" bgcolor="silver">
                     <b>Price</b>
                  </TD>
                  <TD align="center" bgcolor="silver">
                     <b>Published Date</b>
                  </TD>
                  <TD align="center" bgcolor="silver">
                     <b>Description</b>
                  </TD>
               </TR>
               <xsl:for-each select="//book">
                  <TR>
                     <TD>
                        <xsl:value-of select="@id"/>
                     </TD>
                     <TD>
                        <xsl:value-of select="author"/>
```

```
                        </TD>
                        <TD>
                           <xsl:choose>
                              <xsl:when test="self::*[genre = 'Romance']">
                                 <xsl:attribute name="style">background-color:
pink</xsl:attribute>
                              </xsl:when>
                              <xsl:when test="self::*[genre = 'Fantasy']">
                                 <xsl:attribute name="style">background-color:
lightblue</xsl:attribute>
                              </xsl:when>
                              <xsl:otherwise>
                                 <xsl:attribute name="style">background-color:
lightgreen</xsl:attribute>
                              </xsl:otherwise>
                           </xsl:choose>
                           <xsl:value-of select="title"/>
                        </TD>
                        <TD>
                           <xsl:value-of select="genre"/>
                        </TD>
                        <TD>
                           <xsl:if test="price > 6">
                              <xsl:attribute name="bgcolor">cyan</xsl:attribute>
                           </xsl:if>
                           <xsl:value-of select="price"/>
                        </TD>
                        <TD>
                           <xsl:value-of select="publish_date"/>
                        </TD>
                        <TD>
                           <xsl:value-of select="description"/>
                        </TD>
                     </TR>
                  </xsl:for-each>
               </TABLE>
            </BODY>
         </HTML>
      </xsl:template>
</xsl:stylesheet>
```

Sau khi thêm câu:

```
<?xml-stylesheet type="text/xsl" href="catalog.xsl"?>
```

vào đầu hồ sơ **catalog.xml**, double click lên tên file catalog.xml, Internet Explorer sẽ hiển thị kết quả sau:

Book Lovers' Catalog

ID	Author	Title	Genre	Price	Published Date	Description
bk102	Ralls, Kim	Midnight Rain	Fantasy	5.95	2000-12-16	A former architect battles corporate zombies, an evil sorceress, and her own childhood to become queen of the world.
bk107	Thurman,	Splish Splash	Romance	4.95	2000-11-	A deep sea diver finds true love

	Paula				02	twenty thousand leagues beneath the sea.
bk108	Knorr, Stefan	Creepy Crawlies	Horror	4.95	2000-12-06	An anthology of horror stories about roaches, centipedes, scorpions and other insects.
bk109	Kress, Peter	Paradox Lost	Science Fiction	6.95	2000-11-02	After an inadvertant trip through a Heisenberg Uncertainty Device, James Salway discovers the problems of being quantum.
bk110	O'Brien, Tim	Microsoft .NET: The Programming Bible	Computer	36.95	2000-12-09	Microsoft's .NET initiative is explored in detail in this deep programmer's reference.

Bạn có thể tải về catalog.xml và catalog.xsl tại đây.

Dùng nhiều Templates trong một Style Sheet

Trong bài trước, trong mỗi XSL Style Sheet ta thấy vỏn vẹn chỉ có một Template (bảng kẻm in), và nó được áp dụng vào Root Element của tài liệu XML.

Thật ra, XSL cũng cho phép ta dùng nhiều Templates trong một Style Sheet. Có thể bạn cần làm việc ấy vì hai lý do. Thứ nhất, bạn có thể phân chia cách trình bày ra từng phần của tài liệu XML, để dễ debug hay sửa đổi bộ phận nào của Style sheet. Thứ hai, bạn có thể dùng XPath expressions để áp dụng kiểu trình bày nào vào loại dữ liệu nào tùy theo trị số của nó.

Khi một Style Sheet chứa nhiều templates, bạn chỉ định việc áp dụng của chúng vào luận lý trình bày (presentation logic) bằng cách dùng lệnh **apply-templates**. Thông thường, bạn tạo một Template cho Root Element nói là để chế biến cả tài liệu và dùng lệnh **apply-templates** để chế biến những Element nằm bên trong cái top-level template ấy. Những Templates nầy có thể được gọi lúc nào cần, và cái top-level template sẽ xử lý mọi dữ liệu không có Template nào nhắc tới. Tức là nếu Element nào không có template để áp dụng cho nó thì ta dùng cái template tổng quát của Root Element.

Thí dụ như cái Style Sheet sau đây gồm có: một top-level template để áp dụng vào Document (Root) Element, một template cho những Element Product với Attribute UnitPrice có trị số lớn hơn 70, một template cho những Element Product khác, và một template cho những Element Quantity:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:template match="/">
      <HTML>
         <HEAD>
            <TITLE>Northwind Home Page</TITLE>
         </HEAD>
         <BODY>
```

```
            <P>Customer Order</P>
            <P>Order No:
               <xsl:value-of select="Order/@OrderNo"/>
            </P>
            <P>Date:
               <xsl:value-of select="Order/OrderDate"/>
            </P>
            <P>Customer:
               <xsl:value-of select="Order/Customer"/>
            </P>
            <TABLE Border="0">
               <TR>
                  <TD>ProductID</TD>
                  <TD>Product Name</TD>
                  <TD>Price</TD>
                  <TD>Quantity Ordered</TD>
               </TR>
               <xsl:for-each select="Order/Item">
                  <TR>
                     <xsl:apply-templates></xsl:apply-templates>
                  </TR>
               </xsl:for-each>
            </TABLE>
         </BODY>
      </HTML>
   </xsl:template>
   <xsl:template match="Product[@UnitPrice > 70]">
      <TD>
         <xsl:value-of select="@ProductID"/>
      </TD>
      <TD>
         <A>
            <xsl:attribute name="HREF">Products.asp?ProductID=
               <xsl:value-of select="@ProductID"/>
            </xsl:attribute>
            <xsl:value-of select="."/>
         </A>
      </TD>
      <TD>
         <FONT color="red">
            <xsl:value-of select="@UnitPrice"/>
         </FONT>
      </TD>
   </xsl:template>
   <xsl:template match="Product">
      <TD>
         <xsl:value-of select="@ProductID"/>
      </TD>
      <TD>
         <A>
            <xsl:attribute name="HREF">Products.asp?ProductID=
               <xsl:value-of select="@ProductID"/>
            </xsl:attribute>
            <xsl:value-of select="."/>
         </A>
      </TD>
      <TD>
         <xsl:value-of select="@UnitPrice"/>
      </TD>
   </xsl:template>
```

```
   <xsl:template match="Quantity">
      <TD>
         <xsl:value-of select="."/>
      </TD>
   </xsl:template>
</xsl:stylesheet>
```

Khi áp dụng Style Sheet nầy vào cái tài liệu đặt hàng XML, ta sẽ có hồ sơ HTML sau đây:

```
<HTML>
   <HEAD>
      <TITLE>Northwind Home Page</TITLE>
   </HEAD>
   <BODY>
      <P>Customer Order</P>
      <P>Order No: 1047</P>
      <P>Date: 2002-03-26</P>
      <P>Customer: John Costello</P>
      <TABLE Border="0">
         <TR>
            <TD>ProductID</TD>
            <TD>Product Name</TD>
            <TD>Price</TD>
            <TD>Quantity Ordered</TD>
         </TR>
         <TR>
            <TD>1</TD>
            <TD>
               <A HREF="Products.asp?ProductID=1">Chair</A>
            </TD>
            <TD>70</TD>
            <TD>6</TD>
         </TR>
         <TR>
            <TD>2</TD>
            <TD>
               <A HREF="Products.asp?ProductID=2">Desk</A>
            </TD>
            <TD><FONT color="red">250</FONT></TD>
            <TD>1</TD>
         </TR>
      </TABLE>
   </BODY>
</HTML>
```

Phần **BODY** của HTML trên hiển thị như sau:

Customer Order

Order No: 1047

Date: 2002-03-26

Customer: John Costello

ProductID	Product Name	Price	Quantity Ordered

1	Chair	70	6
2	Desk	250	1

Cách áp dụng Style Sheet vào tài liệu XML

Trước khi tiếp tục học thêm các lệnh khác của XSL Style Sheet, ta cần hiểu và biết cách áp dụng một Style Sheet vào một tài liệu XML.

Áp dụng một Style Sheet là một chức năng của một XML parser như MSXML của Internet Explorer. Bạn có thể bảo một XML parser áp dụng một Style Sheet vào một XML bằng cách hoặc là chỉ cần nhét một **processing instruction** vào đầu hồ sơ XML, hoặc là viết một vài dòng code.

Dùng XML parser để hiển thị

Nếu ta lưu trữ XSL Style Sheet của hồ sơ đặt hàng trong một file tên **Order.xsl** thì ta có thể thêm một hàng **processing instruction *xml-stylesheet*** vào đầu hồ sơ đặt hàng XML như sau:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="Order.xsl"?>
<Order OrderNo="1047">
   <OrderDate>2002-03-26</OrderDate>
   <Customer>John Costello</Customer>
   <Item>
      <Product ProductID="1" UnitPrice="70">Chair</Product>
      <Quantity>6</Quantity>
   </Item>
   <Item>
      <Product ProductID="2" UnitPrice="250">Desk</Product>
      <Quantity>1</Quantity>
   </Item>
</Order>
```

Khi một XML parser đọc hồ sơ XML nầy, cái processing instruction *xml-stylesheet* bảo parser áp dụng hồ sơ style sheet **Order.xsl** để transform XML.

Attribute **type** cho biết loại style sheet được áp dụng, hoặc là **XSL style sheet** hoặc là **cascading style sheet (CSS)**, một loại style sheet dùng để chỉ định màu và kiểu chữ. Ở đây nó là XSL style sheet trong dạng text.

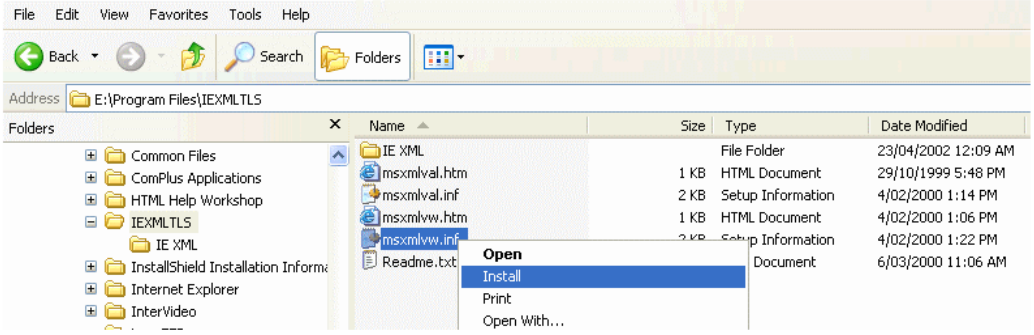
Attribute **href** cho biết tên của file dùng làm Style Sheet, path của tên file ấy có thể là tương đối hay tuyệt đối. Ở đây filename của style sheet là **Order.xsl**, không có path, nên có nghĩa là nó nằm trong cùng một folder với **Order.xml**.

Nếu ta dùng một chương trình trình duyệt như Internet Explorer 5.5 hay 6.0 nó sẽ tự động load Style Sheet để thêm dáng điệu cho tài liệu XML.

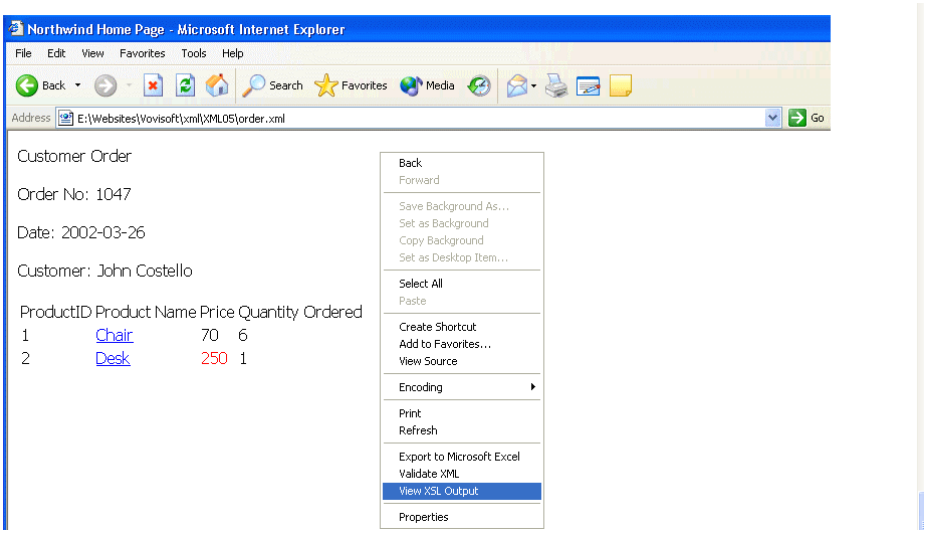
Trong lúc Internet Explorer hiển thị kết quả, nếu bạn dùng Menu Command **View | Source** của browser, bạn sẽ chỉ thấy code của XML, chớ không thấy code HTML như bạn đoán. Muốn xem được code HTML, là kết quả của việc transform XML bằng cách áp dụng XSL bạn cần tải về chương trình công cụ gọi là **Internet Explorer XML/XSL Viewer Tools** từ Microsoft Downloads.

Sau khi Unzip file vừa tải về, bạn right click tên của hai files **msxmlval.inf** và **msxmlvw.inf** rồi chọn **install** để

cài chúng làm Add-ins (những thành phần thêm chức năng vào một chương trình có sẵn) vào chương trình Internet Explorer như trong hình dưới đây.



Bây giờ muốn xem code HTML, bạn right click lên trang Web trong IE rồi chọn command **View XSLOutput** từ PopUpMenu như trong hình dưới đây:



Dùng code để transform với XSL

Cách dùng một ngôn ngữ lập trình để bảo một XML parser chế biến một tài liệu XML sẽ tùy thuộc vào hoàn cảnh. Nếu bạn dùng Microsoft XML parser, một component tên MSXML, trong lập trình thì tài liệu XML sẽ được loaded vào trong một **Document Object Model (XMLDom) object**. Kế đó bạn có thể gọi method **transformNode** để áp dụng một XSL style sheet đã được loaded trước đó vào một XMLDom object khác để chế biến XML.

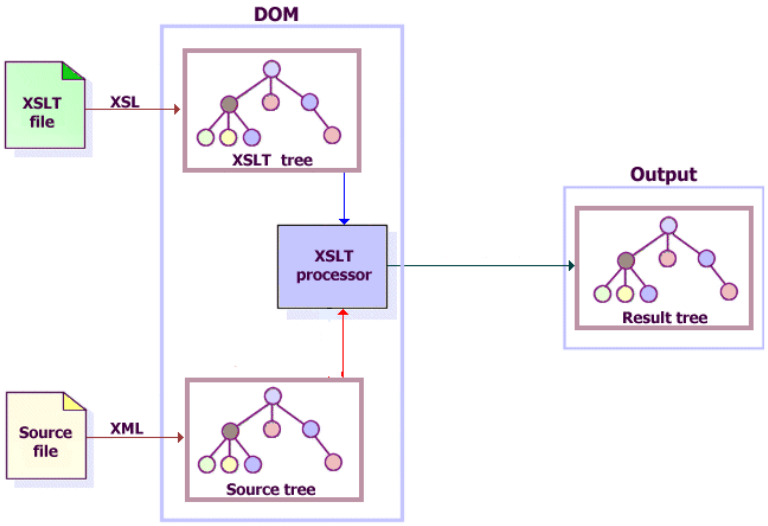
Như trong thí dụ dưới đây, ta dúng hai DOM, một cái để load file **Order.xml**, một cái khác để load **Order.xsl** trong VBScript chạy trên **Active Server Pages (ASP)**:

```
Dim objXML  ' DOM for XML
Dim objXSL  ' DOM for XSL
Dim strResult  ' Resultant document
'Load the XML document.
Set objXML = CreateObject("Microsoft.XMLDom")
objXML.Async = False
objXML.Load "c:\Order.xml"
'Load the XSL style sheet.
Set objXSL = CreateObject("Microsoft.XMLDom")
objXSL.Async = False
objXSL.Load "c:\Order.xsl"
```

```
'Apply the style sheet to XML
strResult = objXML.transformNode(objXSL)
```

Sau khi chạy đoạn code trên, **strResult** sẽ chứa hồ sơ kết quả.

Hình dưới đây minh họa vai trò của **XSLT processor** trong công tác transform một hồ sơ XML dựa vào một XSLT (từ giờ trở đi ta có thể dùng từ **XSLT** thế cho **XSL** cũng được) file:



Ta cũng có thể code bằng JavaScript để chạy trong Browser, thay vì trong WebServer, như cho thấy trong trang Web dưới đây. Nó cũng cho ra cùng một kết quả như khi dùng IE để hiển thị XML trực tiếp.

```
<HTML>
   <HEAD>
      <TITLE>sample</TITLE>
      <SCRIPT language="javascript">
         function init()
           {
             var srcDOM = new ActiveXObject("Msxml2.DOMDocument.4.0");
             srcDOM.async=false;
             srcDOM.load("order.xml");

             var xsltDOM= new ActiveXObject("Msxml2.DOMDOCUMENT.4.0");
             xsltDOM.async = false;
             xsltDOM.load("order.xsl");

             resDOM.innerHTML = srcDOM.transformNode(xsltDOM);
           }
      </SCRIPT>
   </HEAD>
   <BODY onload="init()">
      <div id="resDOM"></div>
   </BODY>
</HTML>
```

Có lẽ bạn hỏi tại sao ta không dùng thẳng XML như phía trên để hiển thị trang Web. Lưu ý là ta có thể dùng kỹ thuật nầy để Transform một XML với XSL rồi hiển thị nó bên trong một **DIV**, tức là một vùng giới hạn bên trong trang Web, chớ không chiếm cả trang Web. Tại đây khi trang Web bắt đầu load (**onload** event), IE gọi **function init()** để transform XML rồi assign kết quả vào **property innerHTML** của **DIV resDOM**.

Có một method khác ta cũng có thể dùng thay cho **transformNode** là **transformNodeToObject**. Sự khác biệt chính giữa hai methods nầy là:

- **transformNode:** Kết quả của method nầy là một tree dưới dạng text string, điển hình là một hồ sơ HTML. Ta có thể cho nó hiển thị trong một browser hay lưu trữ vào một file.
- **transformNodeToObject:** Kết quả của method nầy được để vào trong một object khác, rồi chính object ấy có thể sẽ được chế biến thêm.

Khi ta dùng một trong hai method nói trên, thật ra object nguồn (source object) không cần phải là một hồ sơ đầy đủ. Nó có thể chỉ là một Node của hồ sơ XML. Nếu nó chỉ là một Node thì cái **XSLT processor** xem tập hợp Node ấy, và các Nodes con cháu của nó như một hồ sơ đầy đủ. Tương tự như vậy, một object XSL có thể là một file XSL đầy đủ, hay chỉ là một Node bên trong một file XSL.

Bạn có thể tải về order.xml, order.xsl và trang Web có JavaScript tại đây.

Các ứng dụng của một XML Parser

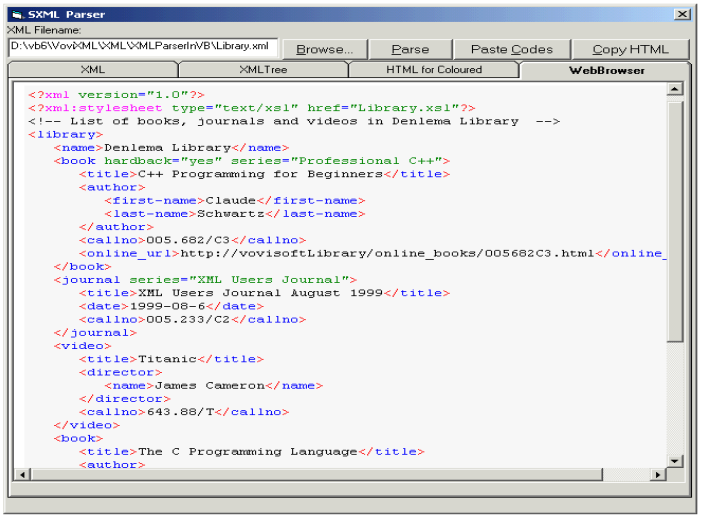
XML càng lúc càng trở nên thịnh hành. Dầu muốn hay không, nếu là software engineer, trước sau gì bạn cũng phải lập trình với XML. Nếu lập trình bằng VB6 bạn có thể dùng **Document Object Model (DOM)** hay **Simple API for XML (SAX)** của Microsoft để giúp đỡ bạn trong công tác parsing (phân tích, sắp đặt) các XML files.

DOM đọc nguyên một XML file rồi parse nó thành một Tree có đẳng cấp trong bộ nhớ, tức là một node cha của Document có những nodes con đại diện cho comments, tags, directives và text (gọi là **XML entities**).

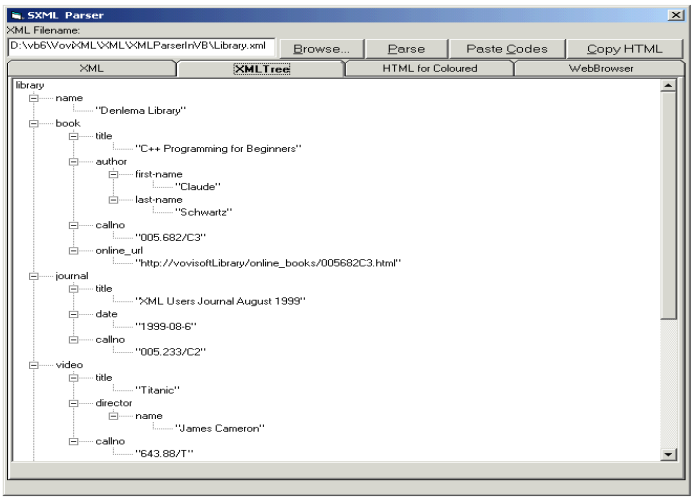
Trong khi đó SAX đọc một XML file và trong khi parse sẽ generate những Events cho hay khi nào nó gặp phải những XML entities. SAX không tạo ra một Tree nào cả, nên các ứng dụng tùy thuộc vào cách ta handle các Events từ SAX. Dĩ nhiên là SAX nhỏ và đơn giản hơn DOM nhiều.

Để không phải tùy thuộc hoàn toàn vào XML parser của người khác và để giúp bạn có ý niệm thực tế về cách làm việc của một XML Parser, trong bài nầy ta sẽ triển khai một XML Parser đơn giản (Simple XML Parser - SXMLParser) hoàn toàn bằng VB6 và áp dụng nó một cách thực tiễn để làm mẫu. SXMLParser tuy nhỏ nhưng có những đặc tính tương tợ như SAX và dĩ nhiên bạn có thể tha hồ sửa đổi, thêm những features tùy ý.

Các áp dụng trước mắt là làm đẹp (Pretty) XML code, thêm màu cho XML content khi hiển thị trong một WebBrowser như trong hình dưới đây:



và tạo một Treeview tượng trưng cho DOM:



Có được source code của XML parser của mình bạn sẽ chiếm ưu thế so với người khác khi thiết kế hay deploy program trên mạng.
Trước khi bàn về program nầy ta hãy ôn lại các qui luật căn bản về một Well-Formed XML.

Well-Formed XML

Mặc dù bạn có thể đặt ra bao nhiêu Tag cũng được, nhưng mỗi trang XML cần phải theo một số qui luật để được xem là Well-Formed (có đầu, có đuôi).
Nếu một trang XML không Well-Formed thì coi như xài không được, không có chương trình xử lý nào sẽ chịu làm việc với dữ liệu bên trong của nó. Do đó một trang XML cần phải theo đúng các qui luật sau đây:

1. Trang XML phải bắt đầu bằng câu tuyên bố XML (XML declaration). Điểm nầy ta có thể bỏ qua được.
2. Mỗi bộ phận, gọi là "**element**" phải nằm giữa một **Tag Pair**.
3. Nếu Tag nào không chứa gì ở giữa thì phải chấm dứt bằng "/>", thí dụ như <BR/> hay <HR/>.
4. Một trang XML phải có một element độc nhất chứa tất cả các elements khác. Đó là root của tree biểu diễn trang XML.
5. Các Tag Pair không được xen kẻ nhau (thí dụ như <name>John Stanmore<address>25 King Street</name></address> là bất hợp lệ vì <address> nằm trong Tag Pair name).

và thêm một vài qui luật về cách dùng các mẫu tự đặc biệt. Ngoài ra các Tag Pair phải đánh vần đúng y như nhau kể cả chữ hoa, chữ thường, (thí dụ: <STUDENT> và </Student> là bất hợp lệ) và tất cả giá trị các **Attributes** đều phải nằm giữa hai ngoặc kép (thí dụ: standalone=yes là bất hợp lệ, phải dùng standalone="yes" mới được.)

Thiết kế SXMLParser

Có một VB6 class chính để lo hầu như hoàn toàn việc parsing một XML file. Sau khi instantiated một Object thuộc **Class clsXMLParser**, ta chỉ cần cho nó tên của XML file là nó bắt đầu công tác parsing ngay.

Như trong hình màu của XML phía trên ta thấy phần chính của XML là từ hàng thứ tư trở đi khi bắt đầu với Open Tag **<library>**. Tương ứng với mỗi Open Tag là có một Close Tag, thí dụ như **</library>**. Bên trong mỗi cặp Tags có thể có những cặp Tags (con) khác.

Một Open Tag có thể chứa nhiều cặp Attributes dưới dạng **Name="Value"**. Lưu ý là Value phải nằm giữa hai dấu ngoặc.

SXMLParser sẽ đi qua từng character một của XML file. Khi đọc xong một Open Tag, thí dụ như:

```
<book hardback="yes" series="Professional C++">
```

SXMLParser sẽ Raise một **StartElement Event** để được handled trong Form chánh bởi **Sub XMLParser_StartElement**. Event nầy cho Form chánh tên của Tag và một collection của các cặp Name="Value" Attributes, thí dụ như Tag **book** đầu tiên chứa **hardback="yes" series="Professional C++"**, chẳng hạn.

Trong Sub XMLParser_StartElement ta làm cùng một lúc ba chuyện:

1. Làm đẹp XML code, tức là các hàng thụt ra, thục vào tùy theo thứ bậc cho dễ đọc.
2. Thêm màu cho HTML file để hiển thị XML code trong WebBrowser
3. Tạo các Nodes trong TreeView

```
Private Sub XMLParser_StartElement(ByVal Name As String, ByVal tagAttributes As
clsAttributes)
   ' A complete Start Element has been processed
   Dim TStr
   ' Build a string of Atributes' Name="Value" pairs
   TStr = BuildAttributeString(tagAttributes)
   ' Display Name Tag in Pretty XML Listbox
   lstXML.AddItem Space(XMLParser.NestedLevel * TabWidth) & "<" & Name & TStr & ">"
   ' Add blue color to the equal sign
   TStr = Replace(TStr, "=", "=")
   ' prepare colour HTML Name tag
   lstHTML.AddItem Space(XMLParser.NestedLevel * TabWidth) & "<Font color=red><</Font>"
_
& "<Font color=blue>" & Name & "" & "<Font color=green>" & TStr & "</Font>" & "<Font
color=red>></Font>"
   ' add a node to the Treeview and save its index in the stack of nested nodes
   If XMLParser.NestedLevel = 0 Then
      ' create the root node
      With XMLTree.Nodes.Add(, , , Name)
         nodeStack(0) = .Index  ' save the node index in stack
         .Expanded = True  ' Expand node
      End With
   Else
      ' create a child node of the higher nested level mode
      With XMLTree.Nodes.Add(nodeStack(XMLParser.NestedLevel - 1), tvwChild, , Name)
         nodeStack(XMLParser.NestedLevel) = .Index  ' save the node index in stack
         .Expanded = True  ' Expand node
      End With
   End If
End Sub
```

Để tái tạo hàng Name="Value" cho collection của các Attributes của một Tag ta dùng **Function BuildAttributeString** như sau:

```
Function BuildAttributeString(ByVal tagAttributes As clsAttributes) As String
   ' Build a string of Atributes' Name="Value" pairs for Element or Instruction
   Dim i, TStr
   Dim attr As clsAttributeItem
   ' Iterate through each Attribute in the collection
   For i = 1 To tagAttributes.Count
      ' refer to i-th attribute
      Set attr = tagAttributes.Item(i)
      ' Start with a space, create string Name="Value"
      TStr = TStr & " " & attr.Name & "=""" & attr.Value & """"
   Next
   BuildAttributeString = TStr  ' Return the resultant string
End Function
```

Dưới đây là danh sách các Events raised bởi SXMLParser để Form chánh xử lý:

```
' Start of parsing
```

```
Event StartDocument()
' End of parsing
Event EndDocument()
' An XML Instruction has been parsed
Event ProcessingInstruction(ByVal Name As String, ByVal tagAttributes As clsAttributes)
' An XML comment has been parsed
Event Comment(ByVal Text As String)
' An open tag as been parsed
Event StartElement(ByVal Name As String, ByVal tagAttributes As clsAttributes)
' A close tag as been parsed
Event EndElement(ByVal Name As String)
' A block of text has been parsed
Event Characters(ByVal Text As String)
' Error encountered while parsing
Event ParseError(ByVal ErrorNo As Integer, ByVal Description As String)
```

Trong khi parsing SXMLParser thay đổi State hay Mode tùy theo trạng thái nó đang tìm kiếm thứ gì, chẳng hạn như character <, >, Attribute Name, Attribute Value, Close Tag .v.v.. Nếu nó khám phá là XML không Well-Formed thì nó sẽ Raise một **ParseError Event** với lý do và chi tiết liên hệ về Error ấy để hiển thị trong Form chánh, giúp User biết cần sửa đổi ở đâu trong XML file.

Danh sách các loại Error mà SXMLParser support được liệt kê dưới đây. Xin lưu ý là có khi Error Message không rõ ràng như ta tưởng tượng vì parser không thông minh như chúng ta.

```
Const cParseEmptyXML = 1
Const cParseNoCommentCloseTag = 2
Const cParseNoValueCloseQuote = 3
Const cParseNoAttributeName = 4
Const cParseNoEqualSign = 5
Const cParseNoAttributeValue = 6
Const cParseNoCDataCloseTag = 7
Const cParseUnknownSymbols = 8
Const cParseNoOpenQuote = 9
Const cParseBadOpenTag = 10
Const cParseBadCloseTag = 11
Const cParseMismatchTagName = 12
Const cParseNoInstructionCloseTag = 13
```

Vì các Tag Pairs cần phải có Tag Names giống nhau hoàn toàn (chữ hoa, chữ thường) và không xen kẻ nhau được, nên ta cần có một Stack để chứa các Tag Names theo đúng đẳng cấp trên dưới. Một Stack là một danh sách theo thứ tự Last-In, First-Out, tức là cái gì mới vào nhất sẽ ra đầu tiên.

Ta thực hiện Stack nầy bằng **Class clsStack**. clsStack chứa các Items thành một String, mà các Items được ngăn cách nhau bởi một vbNullChar (character có ASC value bằng 0). Item mới nhất (Last-In) nằm đầu bên trái của String.

Có ba Functions chánh của Class clsStack là **Push** (để nhét một TagName vào), **Pop** (để lấy TagName mới nhất ra) và **LastIn** (để chỉ xem TagName mới nhất, chớ không lấy ra).

```
Public Sub Push(InItem As String)
   ' Push a Item up the Stack.
   ' Remove any vbNullChars in the Item
   ' Use vbNullChar as the Delimiter
   '
   ' Prefix the Item to the Stack string
   mStacks = Replace(InItem, vbNullChar, "") & vbNullChar & mStacks
   mCount = mCount + 1  ' Increment the Item count
End Sub
```

```
Public Function Pop() As String
   ' Remove and return the LastIn Item in the Stack.
   Dim Pos
   If mCount > 0 Then
      Pos = InStr(mStacks, vbNullChar)  ' Locate vbNullChar
      If Pos > 0 Then
         Pop = Left(mStacks, Pos - 1)  ' Extract the LastIn Item
         mStacks = Mid(mStacks, Pos + 1)  ' Keep the remain of the Stack string
         mCount = mCount - 1  ' decrement the Item Count
      End If
   End If
End Function

Public Function LastIn() As String
   ' View the LastIn Item in the Stack. Leave Stack unchanged
   Dim Pos
   If mCount > 0 Then
      Pos = InStr(mStacks, vbNullChar)  ' Locate vbNullChar
      If Pos > 0 Then
         LastIn = Left(mStacks, Pos - 1)  ' Extract the LastIn Item
      End If
   End If
End Function
```

Bạn có thể download chương trình mẫu **SXMLParser.zip**

Đặc biệt trong program nầy, chỉ cần một click duy nhất lên nút Paste, XML text trong Clipboard sẽ được biến thành HTML code để làm đẹp và hiển thị XML code với màu trong WebBrowser. Ngay sau đó bạn có thể Paste content của Clipboard vào một trang Web.
Dưới đây là listing của **Sub CmdPaste_Click**

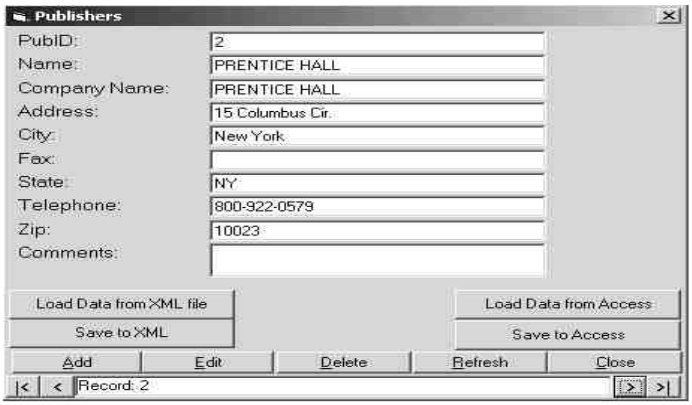
```
Private Sub CmdPaste_Click()
   ' Parse the Clipboard content and copy the resultant colour HTML back to clipboard
   Dim i, TStr
   ' Fetch content of clipboard
   TStr = Clipboard.GetText(vbCFText)
   ' Write it temporarily to "Temp.XML" file in the same folder where this program
resides
   WriteTextFile GetLocalDirectory & "Temp.XML", TStr
   ' Place the XML filename into TextBox txtFilename
   txtFilename.Text = GetLocalDirectory & "Temp.XML"
   ' Emulating User's action of clicking the commandbutton Parse
   cmdParse_Click
   ' If there're something as a result, copy everything from the Listbox lstHTML
   ' except for the first and last line, which contain HTML header/footer.
   ' Select the required lines from the Textbox
   If lstHTML.ListCount > 2 Then
      For i = 1 To lstHTML.ListCount - 2
         lstHTML.Selected(i) = True
      Next
      ' Emulating User's action of clicking the commandbutton Copy
      CmdCopy_Click
   End If
End Sub
```

Hoán chuyển ADO qua XML

Kể từ ActiveX Data Objects version 2.1 (**ADO 2.1**) trở đi, Microsoft ADO engine có thể cho ta XML file dưới dạng **Microsoft XML - Data Schema format**, còn được gọi là **XML Reduced Data Schema**, hay đơn giản hơn là **Reduced Data**. XML Reduced Data Schema nói rõ datatypes và những tính chất tương tợ của schema (tức là default values, tin tức về primary key, .v.v..) từ database và để tin tức nầy trong phần đầu của XML file. Phần sau của XML chứa data trong dạng những **rows**.
Một khi đã có ADO recordset rồi, bạn có thể lưu trử (save) data vào một XML file bằng cách dùng Function **Save** của recordset. ADO 2.1 chỉ cho ta save data vào một XML file. Nhưng ADO 2.5 cho ta convert recordset thành **stream** format. Nếu argument thứ nhất của Function Save là một URL thì Save cho ra data dưới dạng **intrinsic binary format**. Tuy nhiên, nếu ta cho thêm argument thứ nhì là **adPersistXML** flag thì stream được đổi thành một XML stream.
Nếu bạn chưa hề nghe qua danh từ **stream** trước đây, hãy thử tưởng tượng chuyện nầy. Có hai cách để lái buôn dưa hấu giao hàng. Cách thứ nhất họ khiêng dưa hấu từ dưới ghe lên bờ, chất thành một núi nhỏ trên sàn để một chốc sau bạn hàng cho người đến chuyên chở đi. Cách thứ hai, bạn hàng lái xe đến cặp sát bờ sông, một lái buôn đứng dưới ghe ném từng trái dưa hấu lên cho một bạn hàng đứng trên xe chụp rồi chuyển qua cho người khác sắp lên xe nầy hay thảy qua xe khác nếu muốn phân loại dưa hấu lớn nhỏ.
Cách giao hàng thứ nhất giống như save data vào một file. Trong cách giao hàng thứ hai, những trái dưa hấu được ném liên tục bay lên bờ giống như một dòng nước bắn đi, nghĩa đen của chữ stream là dòng nước.
Khi data được chuyển đi dưới dạng một stream, ở đầu nhận có thể xử lý data lập tức, và nhiều khi không cần chứa data nữa. Trong thí dụ nầy, vừa chụp được trái dưa người bạn hàng phải quyết định ngay, nếu dưa hấu lớn thì để lên xe nầy, nếu dưa hấu nhỏ hay nhẹ quá thì thảy qua xe kia.
Trong ADO 2.1, bạn bị bắt buộc phải output stream ra một file, điều nầy có khi phí thì giờ. Cái stream phải được đổi ra Unicode formated text string, spool ra hard disk qua file interface. Rồi nếu bạn cần XML, file ấy phải được loaded và parsed trở lại ra XML stream. ADO 2.5 cho phép bạn viết thẳng kết quả vào một XML **DOM (Document Object Model)** document, khỏi phải save ra file rồi đọc và parse trở lại.

Chương trình mẫu

Bạn có thể download chương trình mẫu **ADOXML.zip** để xem cách save data từ ADO ra XML. Bonus là phần load data từ XML và save ngược lại vào Access Database. Để chạy chương trình ADOXML bạn cần Project | References hai libraries: Microsoft ActiveX Data Objects 2.5 Library và Microsoft Data Binding Collection.



ADO recordset dùng ở đây để biểu diễn data từ table Publishers của BIBLIO.MDB database. Kết quả là một XML file gồm có ba phần:

Phần thứ nhất: data giới thiệu dưới dạng attributes của XML

```
<xml xmlns:s="uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882" xmlns:dt="uuid:C2F41010-65B3-
11d1-A29F-00AA00C14882" xmlns:rs="urn:schemas-microsoft-com:rowset"
xmlns:z="#RowsetSchema">
```

Phần thứ hai: Schema, cắt nghĩa về chính datatype và data structure

```
<s:Schema id="RowsetSchema">
   <s:ElementType name="row" content="eltOnly" rs:updatable="true">
      <s:AttributeType name="PubID" rs:number="1" rs:maydefer="true"
rs:basetable="Publishers" rs:basecolumn="PubID" rs:keycolumn="true">
         <s:datatype dt:type="int" dt:maxLength="4" rs:precision="10"
rs:fixedlength="true"></s:datatype>
      </s:AttributeType>
      <s:AttributeType name="Name" rs:number="2" rs:nullable="true" rs:maydefer="true"
rs:write="true" rs:basetable="Publishers" rs:basecolumn="Name">
         <s:datatype dt:type="string" rs:dbtype="str" dt:maxLength="50"></s:datatype>
      </s:AttributeType>
      <s:AttributeType name="c2" rs:name="Company Name" rs:number="3"
rs:nullable="true" rs:maydefer="true" rs:write="true" rs:basetable="Publishers"
rs:basecolumn="Company Name">
         <s:datatype dt:type="string" rs:dbtype="str" dt:maxLength="255"></s:datatype>
      </s:AttributeType>
      <s:AttributeType name="Address" rs:number="4" rs:nullable="true"
rs:maydefer="true" rs:write="true" rs:basetable="Publishers" rs:basecolumn="Address">
         <s:datatype dt:type="string" rs:dbtype="str" dt:maxLength="50"></s:datatype>
      </s:AttributeType>
      <s:AttributeType name="City" rs:number="5" rs:nullable="true" rs:maydefer="true"
rs:write="true" rs:basetable="Publishers" rs:basecolumn="City">
         <s:datatype dt:type="string" rs:dbtype="str" dt:maxLength="20"></s:datatype>
      </s:AttributeType>
      <s:AttributeType name="Fax" rs:number="6" rs:nullable="true" rs:maydefer="true"
rs:write="true" rs:basetable="Publishers" rs:basecolumn="Fax">
         <s:datatype dt:type="string" rs:dbtype="str" dt:maxLength="15"></s:datatype>
      </s:AttributeType>
      <s:AttributeType name="State" rs:number="7" rs:nullable="true" rs:maydefer="true"
rs:write="true" rs:basetable="Publishers" rs:basecolumn="State">
         <s:datatype dt:type="string" rs:dbtype="str" dt:maxLength="10"></s:datatype>
      </s:AttributeType>
      <s:AttributeType name="Telephone" rs:number="8" rs:nullable="true"
rs:maydefer="true" rs:write="true" rs:basetable="Publishers" rs:basecolumn="Telephone">
         <s:datatype dt:type="string" rs:dbtype="str" dt:maxLength="15"></s:datatype>
      </s:AttributeType>
      <s:AttributeType name="Zip" rs:number="9" rs:nullable="true" rs:maydefer="true"
rs:write="true" rs:basetable="Publishers" rs:basecolumn="Zip">
         <s:datatype dt:type="string" rs:dbtype="str" dt:maxLength="15"></s:datatype>
      </s:AttributeType>
      <s:AttributeType name="Comments" rs:number="10" rs:nullable="true"
rs:maydefer="true" rs:write="true" rs:basetable="Publishers" rs:basecolumn="Comments">
         <s:datatype dt:type="string" rs:dbtype="str" dt:maxLength="1073741824"
rs:long="true"></s:datatype>
      </s:AttributeType>
      <s:extends type="rs:rowbase"></s:extends>
   </s:ElementType>
</s:Schema>
```

Phần thứ ba: data, mỗi datafield value là một attribute value của row

```
<rs:data>
   <z:row PubID="1" Name="SAMS" c2="SAMS" Address="11711 N. College Ave., Ste 140"
City="Carmel" Fax=" " State="IN" Telephone=" " Zip="46032" Comments=" "></z:row>
   <z:row PubID="2" Name="PRENTICE HALL" c2="PRENTICE HALL" Address="15 Columbus Cir."
City="New York" Fax=" " State="NY" Telephone="800-922-0579" Zip="10023" Comments="
"></z:row>
   <z:row PubID="3" Name="M & T" c2="M & T BOOKS" Address=" " City=" " Fax=" " State="
```

```
" Telephone=" " Zip=" " Comments=" "></z:row>
   <z:row PubID="4" Name="MIT" c2="MIT PR" Address="Long Island" City=" " Fax=" "
State="N.Y." Telephone=" " Zip=" " Comments=" "></z:row>
   <z:row PubID="5" Name="MACMILLAN COMPUTER" c2="MACMILLAN COMPUTER PUB" Address="11
W. 42nd St., 3rd flr." City="New York" Fax=" " State="NY" Telephone="212-869-7440"
Zip="10036" Comments=" "></z:row>
   <z:row PubID="6" Name="HIGHTEXT PUBNS" c2="HIGHTEXT PUBNS" Address=" " City=" "
Fax=" " State=" " Telephone=" " Zip=" " Comments=" "></z:row>
   <z:row PubID="7" Name="SPRINGER VERLAG" c2="SPRINGER VERLAG" Address=" " City=" "
Fax=" " State=" " Telephone=" " Zip=" " Comments=" "></z:row>
   <z:row PubID="8" Name="O" REILLY=" c2=" O'REILLY=" Address=" 90="
                City=" Cambridge'=" " State="MA" Telephone=" " Zip="02140" Comments="
"></z:row>
   <z:row PubID="9" Name="ADDISON-WESLEY" c2="ADDISON-WESLEY PUB CO" Address="Rte 128"
City="Reading" Fax="617-964-9460" State="MA" Telephone="617-944-3700" Zip="01867"
Comments=" "></z:row>
   <z:row PubID="10" Name="JOHN WILEY & SONS" c2="JOHN WILEY & SONS" Address="605 Third
Ave" City="New York" Fax="212-850-6088 " State="NY" Telephone="212-850-6000"
Zip="10158" Comments="DATABASES MICROCOMPUTER SOFTWARE PAPERBACK BOOKS - TRADE
TEXTBOOKS - COLLEGE DICTIONARIES, ENCYCLOPEDIAS PERIODICALS
PROFESSIONAL BOOKS
SCIENCE (GENERAL)
BUSINESS
SOCIAL SCIENCES AND SOCIOLOGY
COMPUTER SCIENCE, DATA PROCESSING
ENGINEERING (GENERAL)"></z:row>
   <z:row PubID="11" Name="SINGULAR" c2="SINGULAR PUB GROUP" Address=" " City=" " Fax="
" State=" " Telephone=" " Zip=" " Comments=" "></z:row>
   <z:row PubID="12" Name="Duke Press" c2="Duke Press" Address=" " City=" " Fax=" "
State=" " Telephone=" " Zip=" " Comments=" "></z:row>
   <z:row PubID="13" Name="Oxford University" c2="Oxford University Press" Address=" "
City=" " Fax=" " State=" " Telephone=" " Zip=" " Comments=" "></z:row>
   <z:row PubID="14" Name="Mit Press" c2="Mit Press" Address=" " City=" " Fax=" "
State=" " Telephone=" " Zip=" " Comments=" "></z:row>
   <z:row PubID="15" Name="CAMBRIDGE UNIV" c2="CAMBRIDGE UNIV PR" Address=" " City=" "
Fax=" " State=" " Telephone=" " Zip=" " Comments=" "></z:row>
   <z:row PubID="16" Name="Q E D" c2="Q E D PUB CO" Address=" " City=" " Fax=" "
State=" " Telephone=" " Zip=" " Comments=" "></z:row>
   <z:row PubID="17" Name="Cambridge University" c2="Cambridge University Press"
Address=" " City=" " Fax=" " State=" " Telephone=" " Zip=" " Comments=" "></z:row>
   <z:row PubID="18" Name="WORLD SCIENTIFIC" c2="WORLD SCIENTIFIC PUB CO" Address=" "
City=" " Fax=" " State=" " Telephone=" " Zip=" " Comments=" "></z:row>
   <z:row PubID="19" Name="IDG" c2="IDG BOOKS WORLDWIDE" Address=" " City=" " Fax=" "
State=" " Telephone=" " Zip=" " Comments=" "></z:row>
   <z:row PubID="20" Name="GOWER PUB" c2="GOWER PUB CO" Address=" " City=" " Fax=" "
State=" " Telephone=" " Zip=" " Comments=" "></z:row>
</rs:data>
</xml>
```


Tạo HTML từ XML và XSL

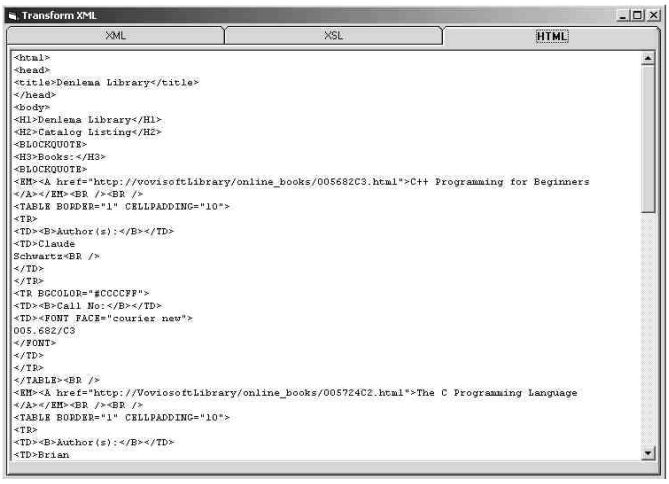
Thông thường, nếu bạn định nghĩa bên trong một XML file rằng nó cần một XSL để display thì Internet Exporer 5.5 sẽ tự transform XML dựa theo XSL và display kết quả trong browser cho bạn. Trong trường hợp ấy nếu bạn View Source của Webpage bạn chỉ đọc được XML source mà thôi. Có thể bạn sẽ thắc mắc sao không thấy cái HTML source là kết quả của quá trình **Tranform** XML.

Trong chương trình VB6 trình bày tại đây bạn sẽ thấy cách dùng **DOM (Document Object Model)** của MSXML để thực hiện công việc tương đương với IE 5.5 và save kết quả HTML thành một Webpage . Webpage nầy có thể

được display, độc lập với XML, bằng bất cứ WebBrowser nào.Thật ra, ActiveX MSXML mà ta dùng cho VB6 là của IE 5.5 khi ta Project | Reference **"Microsoft XML, v3.0"**.

Chương trình mẫu

Bạn có thể download chương trình mẫu **TransformXML.zip** để xem cách thảo chương rất đơn giản.



Phần chính của chương trình nằm trong Sub Form_Load như liệt ra dưới đây. Bạn có thể xem listing và giải thích về cách transform **Library.xml** dựa trên **Library.xsl** trong bài XML, Kỹ thuật tin học nòng cốt trong tương lai.

```
Private Sub Form_Load()
   Dim HTMLCode As String
   ' Need to Project | References "Microsoft XML, v3.0" to use DOM
   Dim myXMLDoc As New MSXML2.DOMDocument30
   Dim myXSLDoc As New MSXML2.DOMDocument30
   ' Need to Project | References "Microsoft Script Runtime" to use
   '   FileSystemObject and TextStream
   Dim Fs As FileSystemObject
   Dim TS As TextStream
   ' Display the XML file in a listbox
   PopulateListBoxFromFile LstXML, "Library.xml", False
   ' Display the XSL file in a listbox
   PopulateListBoxFromFile lstXSL, "Library.xsl", False
   ' Load the XML file
   myXMLDoc.Load App.Path & "\Library.xml"
   ' Load the XSL file
   myXSLDoc.Load App.Path & "\Library.xsl"
   ' Transform XML by XSL to give HTML
   HTMLCode = myXMLDoc.transformNode(myXSLDoc)
   ' Now Write the resultant HTML to file
   ' Create a FileSystem Object
   Set Fs = CreateObject("Scripting.FileSystemObject")
   ' Open TextStream for Output
   Set TS = Fs.OpenTextFile(App.Path & "\Library.htm", ForWriting, False,
TristateUseDefault)
   TS.Write HTMLCode  ' Write the whole HTML string in one stroke
   TS.Close  ' Close the Text Stream
   Set Fs = Nothing  ' Dispose FileSystem Object
   ' Display the HTML file in a listbox
   PopulateListBoxFromFile lstHTML, "Library.htm", False
End Sub
```

Dùng DOM để display XML trong TreeView

Internet Explorer 5.0 cho ta **Document Object Model (DOM)** ActiveX gọi là **MSXML.DLL** mà ta có thể dùng trong VB6. Đầu tiên là **Microsoft XML, version 2.0**, tiếp theo đó là **Microsoft XML, v2.6** và mới nhất là **Microsoft XML, v3.0**. Cả ba DLL nầy đều có trong danh sách các References mà ta có thể include khi dùng IDE Menu command **Project | References**.
Khi ta **Load** một XML file vào DOM, nó tự động **parse** XML data để build một Tree gồm nhiều nodes với thứ bậc cha, con bên trong. Dựa theo đó ta có thể display cái DOM Tree ấy trong một TreeView để có thể hình dung được cấu trúc của XML data.

Trong thí dụ dưới đây, ta Load một XML file tên **people.xml** vào DOM. XML file nầy còn có một Data Type Definition file tên **people.dtd**. Khi DOM load XML file, ta có thể dặn nó kiểm (validate) xem XML data có theo đúng tiêu chuẩn đòi hỏi trong dtd file.
Content của **people.xml** như sau, lưu ý hàng thứ hai nhắc đến **people.dtd** mà DOM sẽ dùng để validate data trong XML file:

```
<?xml version="1.0"?>
<!DOCTYPE PEOPLE SYSTEM "people.dtd">
<PEOPLE>
        <PERSON PERSONID="p1">
                <NAME>Peter Greenway</NAME>
                <ADDRESS>234 King St, Newtown, NSW, Australia</ADDRESS>
                <TEL>(612) 97463534</TEL>
                <FAX>(612) 97463535</FAX>
                <EMAIL>pgreenway@ozemail.com.au</EMAIL>
        </PERSON>
        <PERSON PERSONID="p2">
                <NAME>Sue Williams</NAME>
                <ADDRESS>72/324 John St, Cabramatta, NSW, Australia</ADDRESS>
                <TEL>(612) 9745 2263</TEL>
                <FAX>(612) 9745 2264</FAX>
                <EMAIL>swilliams@bigpond.com.au</EMAIL>
        </PERSON>
        <PERSON PERSONID="p3">
                <NAME>Helen Clark</NAME>
                <ADDRESS>74 GreenHill Rd, Wayville, SA, Australia</ADDRESS>
                <TEL>(618) 9756 3635</TEL>
                <FAX>(618) 9756 3636</FAX>
                <EMAIL>hclark@tgp.com.au</EMAIL>
        </PERSON>
        <PERSON PERSONID="p4">
                <NAME>Martin Howard</NAME>
                <ADDRESS>652 Broadbeach Drive, St Kilda, Melbourne, Australia</ADDRESS>
                <TEL>(613) 9756 2312</TEL>
                <FAX>(613) 9756 2313</FAX>
                <EMAIL>mhoward@island.net.au</EMAIL>
        </PERSON>
        <PERSON PERSONID="p5">
                <NAME>Pam Rose</NAME>
                <ADDRESS>24/274 Stancey St, Bankstown, NSW, Australia</ADDRESS>
                <TEL>(612) 9867 9821</TEL>
                <FAX>(612) 9867 9822</FAX>
                <EMAIL>prose@globalfreeway.com.au</EMAIL>
        </PERSON>
        <PERSON PERSONID="p6">
                <NAME>Le Duc Hong</NAME>
```

```
                <ADDRESS>3 Rawson St, Epping, NSW,Australia</ADDRESS>
                <TEL>(612) 9783 1442</TEL>
                <FAX>(612) 9783 1445</FAX>
                <EMAIL>ldhong@dingoblue.com.au</EMAIL>
               </PERSON>
        <PERSON PERSONID="p7">
                <NAME>Âu Địch Xương</NAME>
                <ADDRESS>435 Trần Hưng Đạo, Vĩnh Long , Việt Nam</ADDRESS>
                <TEL>847 74847</TEL>
                <FAX>847 9682</FAX>
                <EMAIL>dixonau@vovisoft.com</EMAIL>
        </PERSON>
        <PERSON PERSONID="p8">
                <NAME>Lý Phúc Hiếu</NAME>
                <ADDRESS>234 Lý Công Uẩn, Saigon, Việt Nam </ADDRESS>
                <TEL>827 3746</TEL>
                <FAX>827 4645</FAX>
                <EMAIL>lyhieu@vnn.vn</EMAIL>
        </PERSON>
</PEOPLE>
```

Content của **people.dtd** như sau:

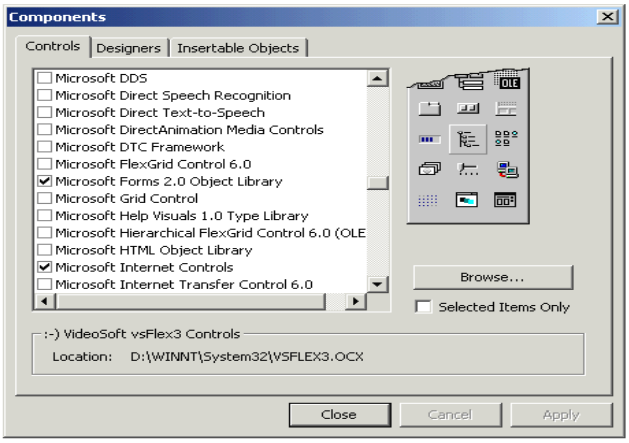
```
<!ELEMENT PEOPLE ( PERSON+ ) >
<!ELEMENT PERSON ( NAME, ADDRESS, TEL, FAX, EMAIL ) >
<!ATTLIST PERSON PERSONID ID #REQUIRED>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT ADDRESS ( #PCDATA ) >
<!ELEMENT TEL ( #PCDATA ) >
<!ELEMENT FAX ( #PCDATA ) >
<!ELEMENT EMAIL ( #PCDATA ) >
```

Trong file **people.dtd** phía trên ta có:

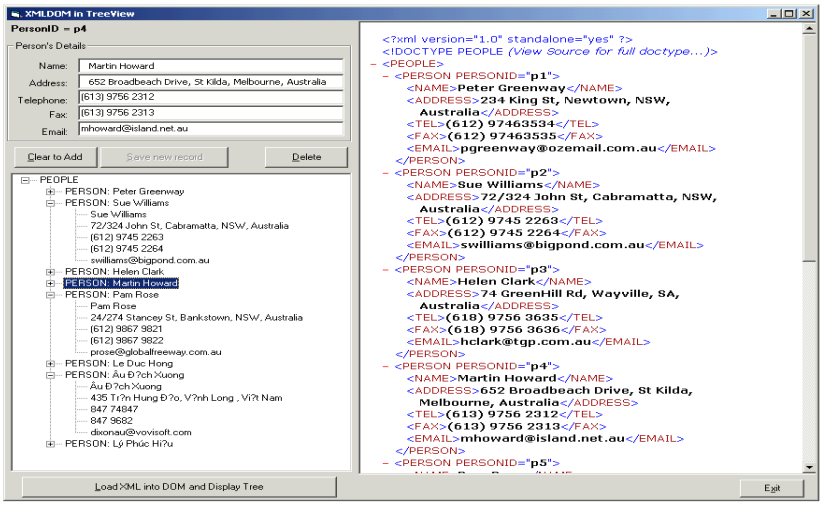
1. Hàng thứ nhất nói rằng cái **root Node** (Node gốc) tên là **PEOPLE**. Nó có một hay nhiều Nodes con tên **PERSON**.
2. Hàng thứ nhì nói mỗi Node **PERSON** có những Nodes con tên **NAME, ADDRESS, TEL, FAX** và **EMAIL**.
3. Các hàng còn lại cho biết mỗi Node **NAME, ADDRESS, TEL, FAX, EMAIL** đều chứa text string.

Việc đầu tiên khi chạy program là bạn click nút **Load XML into DOM and Display Tree**. Đợi một chút xíu, Tree của XML sẽ hiện ra trong TreeView. Đồng thời XML data cũng được displayed trong WebBrowser phía bên phải. Sau đó, mỗi lần bạn click lên dấu**+** hay **-** bên trái tên của một người trong TreeView, chi tiết của người đó sẽ được display trong các TextBox phía trên.
Sau khi selected một Person, bạn có thể Delete tên đó bằng cách click nút **Delete**. Ngoài ra bạn cũng có thể thêm tên một người bằng cách click nút **Clear to Add**, điền các chi tiết của Person vào các TextBoxes rồi click **Save new record**. Mỗi lần bạn Delete một Person hay Add một New Person, program tự động Save kết quả xuống XML file.

Hai TextBoxes txtName và txtAddress không phải là TextBoxes thông thường nhưng là TextBoxes của ActiveX Form2. Các Controls của Form2 có thể display chữ Việt bằng Unicode. Do đó nếu bạn Click lên tên **Lý Phúc Hiếu** chẳng hạn, bạn sẽ thấy tên và địa chỉ được display trong Font Tahoma có dấu đầy đủ của chữ Việt. Muốn có các Controls của Form2 để display chữ Việt bạn dùng menu command **Project | Components** để popup Components Dialog, kế đó click **Microsoft Forms 2.0 Object Library** như trong hình dưới đây:

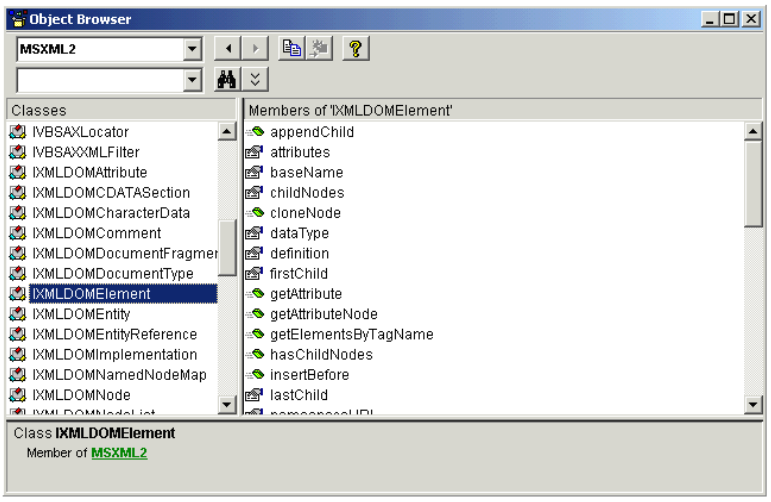


Bạn có thể thử viết thêm code để Edit các chi tiết của một Person có sẵn.



Bạn có thể download chương trình mẫu **DOMTree.zip** để chạy thử.

Để biết thêm các Properties và Methods của các Classes trong MSXML, từ trong VB6 IDE bạn press **F2** để display **Object Browser**. Khi Object Browser Dialog hiện ra, chọn **MSXML2** từ ComboBox phía trên đang display **<All Libraries>**, kế đó chọn một class, thí dụ như **IXMLDOMElement** từ ListBox bên trái, chi tiết của selected Class sẽ được displayed trong ListBox bên phải như trong hình dưới đây:



Dùng DOM để display XML thành nhiều tầng trong TreeView

I nternet Explorer 5.0 cho ta **Document Object Model (DOM)** ActiveX gọi là **MSXML.DLL** mà ta có thể dùng trong VB6. Đầu tiên là **Microsoft XML, version 2.0**, tiếp theo đó là **Microsoft XML, v2.6** và mới nhất là **Microsoft XML, v3.0**. Cả ba DLL nầy đều có trong danh sách các References mà ta có thể include khi dùng IDE Menu command **Project | References**.
Khi ta **Load** một XML file vào DOM, nó tự động **parse** XML data để build một Tree gồm nhiều nodes với thứ bậc cha, con bên trong. Dựa theo đó ta có thể display cái DOM Tree ấy trong một TreeView để có thể hình dung được cấu trúc của XML data.

Trong thí dụ dưới đây, ta Load một XML file tên **Library.xml** vào DOM. XML file nầy còn có một Schema file tên **LibrarySchema.xml**. Khi DOM load XML file, ta có thể dặn nó kiểm (validate) xem XML data có theo đúng tiêu chuẩn đòi hỏi trong Schema file.
Content của **Library.xml** như sau, lưu ý hàng thứ 7 nhắc đến **LibrarySchema.xml** mà DOM sẽ dùng để validate data trong XML file:

```
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="Library.xsl"?>
<!-- Copyright 2000 Wattle Software http://XMLwriter.net/  -->
<library xmlns="x-schema:LibrarySchema.xml">
   <!-- declare the Schema which defines this document  -->
   <name>Northmead Local Library</name>
   <book hardback="yes" availableforloan="no">
      <title>C++ Programming for Beginners</title>
      <author>
         <first-name>Claude</first-name>
         <last-name>Schwartz</last-name>
      </author>
      <callno>005.133/C</callno>
      <online_url>http://library/online_books/005133C.html</online_url>
   </book>
   <journal series="XML Users Journal">
      <title>XML Users Journal August 1999</title>
      <date>1999-08-01</date>
      <callno>005.133/C</callno>
   </journal>
   <video>
      <title>Titanic</title>
      <director>
         <name>James Cameron</name>
      </director>
      <callno>643.11/T</callno>
   </video>
   <book>
      <title>The C Programming Language</title>
      <author>
         <first-name>Brian</first-name>
         <last-name>Kernighan</last-name>
      </author>
      <author>
         <first-name>Dennis</first-name>
         <last-name>Ritchie</last-name>
      </author>
      <callno>005.133/C2</callno>
```

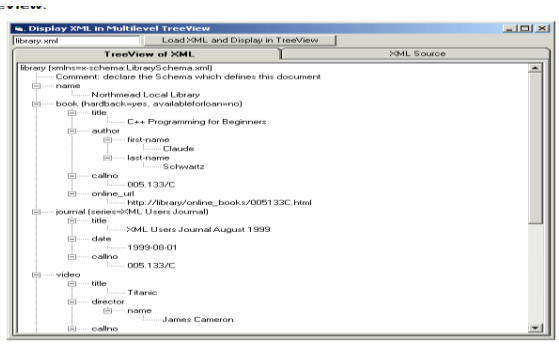
```
      <online_url>http://library/online_books/005133C2.html</online_url>
   </book>
</library>
```

Content của **LibrarySchema.xml** như sau:

```
<?xml version="1.0"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-
com:datatypes">
   <!-- Copyright 2000 Wattle Software http://XMLwriter.net/ This Schema is based on
XML-Schema support found in Microsoft Internet Explorer 5.  -->
   <!-- first we need to declare all elements that will appear only as child elements
-->
   <ElementType name="first-name" content="textOnly"></ElementType>
   <ElementType name="last-name" content="textOnly"></ElementType>
   <ElementType name="name" content="textOnly"></ElementType>
   <ElementType name="price" content="textOnly" dt:type="fixed.14.4"></ElementType>
   <ElementType name="title" content="textOnly" dt:type="string"></ElementType>
   <ElementType name="artist" content="textOnly" dt:type="string"></ElementType>
   <ElementType name="callno" content="textOnly" dt:type="string"></ElementType>
   <ElementType name="date" content="textOnly" dt:type="date"></ElementType>
   <ElementType name="online_url" content="textOnly" dt:type="string"></ElementType>
   <!-- and all attributes too  -->
   <AttributeType name="hardback" dt:type="string"></AttributeType>
   <AttributeType name="availableforloan" dt:type="string"></AttributeType>
   <AttributeType name="series" dt:type="string"></AttributeType>
   <!-- Now we can define the more interesting elements (i.e. those that can have
children )  -->
   <ElementType name="author" content="eltOnly" order="one">
      <!-- An author can contain EITHER: a name, or a sequence of first-name then last-
name  -->
      <group order="seq">
         <element type="name"></element>
      </group>
      <group order="seq">
         <element type="first-name"></element>
         <element type="last-name"></element>
      </group>
   </ElementType>
   <ElementType name="director" content="eltOnly" order="one">
      <!-- A director can contain EITHER: a name, or a sequence of first-name then
last-name  -->
      <group order="seq">
         <element type="name"></element>
      </group>
      <group order="seq">
         <element type="first-name"></element>
         <element type="last-name"></element>
      </group>
   </ElementType>
   <ElementType name="video" content="eltOnly">
      <element type="title"></element>
      <element type="director"></element>
      <element type="callno"></element>
   </ElementType>
   <ElementType name="cd" content="eltOnly">
      <element type="title"></element>
      <element type="artist"></element>
      <element type="callno"></element>
```

```
   </ElementType>
   <ElementType name="journal" content="eltOnly">
      <!-- declare series as an optional attribute of journal  -->
      <attribute type="series" required="no"></attribute>
      <element type="title"></element>
      <element type="date"></element>
      <element type="callno"></element>
   </ElementType>
   <ElementType name="book" content="eltOnly">
      <!-- declare hardback as an optional attribute of journal with default value of
"no"  -->
      <attribute type="hardback" default="no"></attribute>
      <attribute type="availableforloan" default="yes"></attribute>
      <element type="title"></element>
      <!-- allow for multiple authors with maxOccurs  -->
      <element type="author" maxOccurs="*"></element>
      <element type="callno"></element>
      <!-- allow for ONE optional URL  -->
      <element type="online_url" minOccurs="0" maxOccurs="1"></element>
   </ElementType>
   <ElementType name="library" content="eltOnly">
      <!-- the library name comes first  -->
      <element type="name"></element>
      <!-- followed by a collection of books, videos and cds  -->
      <group order="many">
         <element type="book" maxOccurs="*"></element>
         <element type="journal" maxOccurs="*"></element>
         <element type="video" maxOccurs="*"></element>
         <element type="cd" maxOccurs="*"></element>
      </group>
   </ElementType>
</Schema>
```

Việc đầu tiên khi chạy program là bạn click nút **Load XML and Display in TreeView**. Đợi một chút xíu, Tree của XML sẽ hiện ra trong TreeView.
Đồng thời Content của XML file cũng được loaded vào ListBox **lstXMLSource** và bạn sẽ thấy nó nếu bạn click Tab **XML Source**. Dĩ nhiên bạn có thể display bất cứ một XML file nào nếu bạn để nó vào folder của program và enter Filename của nó vào TextBox **txtXMLFileName** trước khi click nút **Load XML and Display in TreeView**.



Trong program nầy ta dùng Object **IXMLDOMNode**, thay vì Object **IXMLDOMElement** để lần lượt đi qua mọi nodes của XML DOM. Program gọi **Sub AddNode** để bỏ các Nodes vào TreeView. Đặc biệt là AddNode gọi chính nó ở bên trong Sub AddNode. Kỹ thuật nầy gọi là **recursive**, mà ta thường lấy dùng trong những cấu trúc giống như nhánh cây, khi chính một **Con** lại có nhiều **Con** khác. Listing của Sub AddNode như sau:

```
Private Sub AddNode(ByRef oElem As IXMLDOMNode, Optional ByRef oTreeNode As Node)
   ' Add a Node to the TreeView
   Dim oNewNode As Node
```

```
   Dim oNodeList As IXMLDOMNodeList
   Dim i As Long
   ' Create the new node
   If oTreeNode Is Nothing Then
      ' Go through here when creating the top level nodes, i.e. childNodes of root node
      Set oNewNode = TreeView.Nodes.Add
   Else
      Set oNewNode = TreeView.Nodes.Add(oTreeNode, tvwChild)
   End If
   ' Expand TreeView node
   oNewNode.Expanded = True
   ' Prepare the Text for the TreeView Node
   If oElem.nodeType = NODE_ELEMENT Then
      ' Element Node type. Use Node name and Attribute values
      oNewNode.Text = BuildNodeLabel(oElem)
   ElseIf (oElem.nodeType = NODE_TEXT) Then
      ' Last Node in the branch. Use Text
      oNewNode.Text = oElem.Text
   ElseIf (oElem.nodeType = NODE_COMMENT) Then
      ' Comment Node. Display the comment
      oNewNode.Text = "Comment:" & oElem.Text
   Else
      ' Display Nodename as default
      oNewNode.Text = oElem.nodeName
   End If
   ' process the childNodes which form a NodeList
   Set oNodeList = oElem.childNodes
   ' Iterate through each childNode
   For i = 0 To oNodeList.length - 1
      ' Recursively call AddNode to add more nodes as children of oNewNode,
      ' treating AddNode just like another Sub
      AddNode oNodeList.Item(i), oNewNode
   Next
End Sub
```

Có ba loại Nodes ta xử lý ở đây: NODE_ELEMENT, NODE_TEXT và NODE_COMMENT. Element Node thì có Node , Attributes và Con. Text Node và Comment Node thì chỉ có text.

Bạn có thể download chương trình mẫu **XMLTreeDOM.zip** để chạy thử.

Để biết thêm các Properties và Methods của các Classes trong MSXML, từ trong VB6 IDE bạn press **F2** để display **Object Browser**. Khi Object Browser Dialog hiện ra, chọn **MSXML2** từ ComboBox phía trên đang display **<All Libraries>**, kế đó chọn một class, thí dụ như **IXMLDOMNode** từ ListBox bên trái, chi tiết của selected Class sẽ được displayed trong ListBox bên phải như trong hình dưới đây: