25. XML

XML is a meta-language for communicating data between disparate technologies. That is, we can create new markup languages using XML to communicate, in this case, financial data, from one application to another. Many segments of the financial industry have already done this specifically for their own domains.

XML allows us to represent the contextual meaning of the data we wish to describe and transmit. This is done through the definition of customized tags. As long as the application that sends an XML message and the application that receives it agree on what these tags mean, they can communicate and exchange data.

25.1 WELL-FORMED XML DOCUMENTS

Every XML document must be well-formed. That is, it must follow all of the structural rules for XML. Programs that read XML, called parsers, will reject any message that does not follow the structural rules for being well formed. Among the most important rules are that XML is case sensitive, and that unclosed tags and overlapping tags are not permitted.

Every start tag must have an end tag. The start tag begins an enclosed area of text, known as an item, according to the tag name. <Ticker> is a start tag. </Ticker> is an end tag. The element, defined by a tag, ends with the end tag. XML tags may also include one of a list of attributes consisting of an attribute name and an attribute value. A tag that opens inside another tag must close before the containing tag closes. Put differently, the structure of an XML document must be strictly hierarchical. But, just because an XML document is well formed, does not mean it is valid.

25.2 VALID XML DOCUMENTS

When tags in a well-formed XML document are queried for their meanings, we say the document is being validated. A valid XML message means that both the sending and receiving parties are able correctly identify the document's content according to an agreed upon set of tag definitions.

We will use an XML Schema to validate an XML stream that is shared between the two parts of our application. Financial organizations publish schemas that outline the format of XML documents according to their protocol. Trading firms that wish to exchange data can then build their XML documents according to these agreed-upon schemas so their conterparts will understand the messages.

Here is a simple XML document that contains data about a trade.

Trade.xml

```
<Trade>
   <TradeID>10FI58B6034</TradeID>
   <Exchange Acronym="CME" />
   <Contract Symbol="ES" Expiry="Sep06"/>
   <Price>110525</Price>
   <Quantity>10</Quantity>
   <Time>26 Jul 2006 11:05:01.1452</Time>
   <Account>00123</Account>
</Trade>
```

25.3 XML SCHEMA DOCUMENTS

XML Schemas are documents that use XML Schema definition (XSD) language to define and validate XML documents. For example, an OTC derivatives trade represented in XML can be validated with an XML Schema before it is sent. This validation verifies that all of the required data is present, is in the proper order and data type. This ensures that the recipient of the trade will be able to interpret the data correctly when it is received.

25.4 PARSERS

As we discussed previously, an XML message's structure should validated against an XSD, XML Schema Definition. This is done through the use of a parser, a program that actually reads the data and conducts the validation. A parser that has access to a XSD guarantees that all the proper elements and attributes are present in an XML document. As with XML itself, the rules for validation are very strict.

XML validation is order sensitive and elements must appear in the same order as they are specified in the XSD. Additional fields may not be added without first defining them in the XSD or in the XML message itself.

25.5 SAMPLE CODE: TRADES.XSD

Given the above XML document, we can generate an XML Schema using Microsoft's XML Schema definition tool (xsd.exe). This tool, though not without its shortcomings will generated an XSD file against which subsequent XML data can be validated. In the command window just type xsd, the path and the file name.

Fig: Chap25_01.tif

```
C:\>xsd ./Trade.xml
```

The XSD file generated by this tool on my machine was not correct. It would not validate the XML file used to create the XML Schema itself. However, by simply copying and pasting the italicized code shown into the proper location in sequence, it worked just fine.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="NewDataSet" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="Trade">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="TradeID" type="xs:string" minOccurs="0" />
        <xs:element name="Exchange" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="Acronym" type="xs:string" />
          </xs:complexType>
        </xs:element>
        <xs:element name="Contract" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="Symbol" type="xs:string" />
            <xs:attribute name="Expiry" type="xs:string" />
          </xs:complexType>
        </xs:element>
        <xs:element name="Price" type="xs:string" minOccurs="0" />
        <xs:element name="Quantity" type="xs:string" minOccurs="0" />
```

```
        <xs:element name="Time" type="xs:string" minOccurs="0" />
        <xs:element name="Account" type="xs:string" minOccurs="0" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="NewDataSet" msdata:IsDataSet="true">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element ref="Trade" />
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

In these simple examples shown, we will see how to write and read an XML document and validate it against the XML Schema.

25.6 SAMPLE CODE: XMLWRITER_EXAMPLE

```
Imports System.Xml
Imports System.Text

Public Sub Main()

     Dim ^m_Writer As New XmlTextWriter( "C:\\Trade.xml", Encoding.ASCII )

     m_Writer.WriteStartDocument()
     m_Writer.WriteDocType( "Trade", Nothing, Nothing, Nothing )

     m_Writer.WriteStartElement( "Trade" )
     m_Writer.WriteElementString( "TradeID", "10FI58B6034" )

     m_Writer.WriteStartElement( "Exchange" )
     m_Writer.WriteAttributeString( "Acronym", "CME" )
     m_Writer.WriteEndElement()

     m_Writer.WriteStartElement( "Contract" )
     m_Writer.WriteAttributeString( "Symbol", "ES" )
     m_Writer.WriteAttributeString( "Expiry", "Sep06" )
     m_Writer.WriteEndElement()

     m_Writer.WriteElementString( "Price", "110525" )
     m_Writer.WriteElementString( "Quantity", "10" )
     m_Writer.WriteElementString( "Time", "26 Jul 2006 11:05:01.1452" )
     m_Writer.WriteElementString( "Account", "00123" )

     m_Writer.WriteEndElement()
     m_Writer.Formatting = Formatting.Indented

     m_Writer.Flush()
     m_Writer.Close()

End Sub
```

25.7 SAMPLE CODE: XMLREADER_EXAMPLE

```
Imports System.io
Imports System.Xml
Public Class Form1

Private Sub Button1_Click(…) Handles Button1.Click

     ‘ Create XML document as above.

End Sub

Private Sub Button2_Click(…) Handles Button2.Click
     Dim m_Settings As New XmlReaderSettings
     m_Settings.ProhibitDtd = False
     m_Settings.ValidationType = ValidationType.None
     Dim m_reader As XmlReader = XmlReader.Create("C:\Trade.xml", m_Settings)

     While m_reader.Read

          Select Case m_reader.NodeType
               Case XmlNodeType.Element
               Case XmlNodeType.Text
                    TextBox6.Text += m_reader.Value + vbCrLf
               Case XmlNodeType.EndElement
          End Select
     End While

End Sub
End Class
```

The Financial Products Markup Language (FpML) is a freely licensed XML protocol for trading complex over-the-counter financial derivative instruments, including equity, interest rate and foreign exchange derivatives such as options, spots, forwards, swaps, and swaptions. Eventually, it is hoped that FpML will automate the flow of information for electronic trading and confirmations in all the types of negotiated OTC derivatives.

Let’s take a look at a sample FpML message taken from the FpML Version 2.0 documentation, which can be found at www.FpML.org. As you will see, this document contains the information about a forward rate agreement trade. In some areas we have abbreviated less interesting content.

On May 14, 1991, ABN AMRO Bank and Midland Bank entered a into a forward rate agreement in which ABN AMRO was the seller of the contract and Midland was the buyer. The terms of the contract are as follows:

- Effective Date: 01/07/1991
- Termination Date: 01/17/1992
- Notional Amount: CHF 25,000,000
- Fixed Rate: 4.00%
- Day Count Fraction: Actual / 360

Here is an XML representation of this OTC trade of a forward rate agreement using the FpML protocol:

```
<?xml version="1.0" ?>
<FpML version="2-0" BusinessCenterSchemeDefault=http://www.fpml.org/…>
<trade>
   <tradeHeader>
      <partyTradeIdentifier>
         <partyReference href="#MIDLAND" />
         <tradeId tradeIdScheme="http://www.hsbc.com/…>123</tradeId>
      </partyTradeIdentifier>
      <partyTradeIdentifier>
         <partyReference href="#ABNAMRO" />
         <tradeId tradeIdScheme="http://www.abnamro.com/…>456</tradeId>
      </partyTradeIdentifier>
      <tradeDate>1991-05-14</tradeDate>
   </tradeHeader>
   <fra>
      <buyerPartyReference href="#MIDLAND" />
      <sellerPartyReference href="#ABNAMRO" />
      <adjustedEffectiveDate id="resetDate">1991-07-17</adjustedEffectiveDate>
      <adjustedTerminationDate>1992-01-17</adjustedTerminationDate>
      <paymentDate>
         <unadjustedDate>1991-07-17</unadjustedDate>
         <dateAdjustments>
            <businessDayConvention>FOLLOWING</businessDayConvention>
            <businessCenters>
                  <businessCenter>CHZU</businessCenter>
            </businessCenters>
         </dateAdjustments>
      </paymentDate>
      <fixingDateOffset>
         <periodMultiplier>-2</periodMultiplier>
         <period>D</period>
         <dayType>Business</dayType>
         <businessDayConvention>NONE</businessDayConvention>
         <businessCenters>
            <businessCenter>GBLO</businessCenter>
         </businessCenters>
         <dateRelativeTo href="#resetDate">ResetDate</dateRelativeTo>
      </fixingDateOffset>
      <dayCountFraction>ACT/360</dayCountFraction>
      <calculationPeriodNumberOfDays>184</calculationPeriodNumberOfDays>
      <notional>
         <currency>CHF</currency>
         <amount>25000000.00</amount>
      </notional>
      <fixedRate>0.04</fixedRate>
      <floatingRateIndex>CHF-LIBOR-BBA</floatingRateIndex>
      <indexTenor>
         <periodMultiplier>6</periodMultiplier>
         <period>M</period>
      </indexTenor>
      <fraDiscounting>true</fraDiscounting>
   </fra>
   <party id="MIDLAND">
      <partyId>MIDLGB22</partyId>
   </party>
   <party id="ABNAMRO">
```

```
      <partyId>ABNANL2A</partyId>
   </party>
</trade>
</FpML>
```

Although this XML document is quite lengthy, you should be able to not only read it, but based upon what we learned in the previous chapter, also be able to understand the underlying structure, how it may be created in .NET and how it could be sent to an IP address.

25.8 SUMMARY

Performance-wise, XML documents are large, relative to ones with binary formats and they use more bandwidth, more storage space, and require more processor time for compression. Furthermore, parsing XML documents is generally a slower process and requires more memory. Good application design can help avoid some of these problems.

As an alternative to using the XmlReader and XmlWriter classes, we can, given an XML Schema, run the XML Schema Definition (xsd.exe) tool to generate classes that, when serialized, will conform to the XML Schema. As we will see in a later chapter, we can use an XmlSerializer object when the XML document adheres to an XML Schema.

Chapter Problems

1.) What is a meta-language?
2.) What is the difference between a well formed XML message and a valid one?
3.) What is FMML and how is it different from XML?
4.) What is a DTD?
5.) What objects are contained in the System.Net and System.XML namespaces?