



# INTRODUCTION TO WEBSERVICES

---

## XML AND E-COMMERCE

---

### ABSTRACT

---

The purpose of this paper is not so much to explain extensible markup language (XML) as it is to explain why XML is so important to electronic commerce.

# INTRODUCTION TO WEBSERVICES

## XML AND E-COMMERCE

---

### INFORMATION IN CONTEXT

---

The purpose of this paper is not so much to explain extensible markup language (XML) as it is to explain why XML is so important to electronic commerce. Even readers who are experienced with XML may want to scan this paper.

Before we begin, however, it is important to introduce a critical concept of *information context*. Table 1 contains an excerpt from a sample balance sheet. In this excerpt we can read that the company had as part of its current assets \$447 million in marketable securities in 1997. We know this because the number 447 is placed in visual context.

(Dollars in millions)

	1997	1996
Current assets		
Cash and cash equivalents	\$7,106	\$7,687
Marketable securities	447	450
Notes and accounts receivable	16,850	16,515

Table 1 Excerpt from a sample balance sheet

We know that the number 447 represents marketable securities because it is on the same row as the phrase “marketable securities”. We know that it is for the year 1997 because it is under the header “1997”. There are more subtle contexts at work here. The label “Dollars in millions” tells us that the number 447 actually represents 447,000,000. Furthermore, we know that marketable securities are a subset of current assets because the label is indented. We also know that the number 447 represents dollars because the row just above has a dollar sign with \$7,106.

(Dollars in millions)

	1997	1996
Current assets		
Cash and cash equivalents	\$ 7,106	\$ 7,687
Marketable securities	447	450
Notes and accounts receivable	16,850	16,515

Figure 1 The table provides visual context

For centuries, graphic designers and publishers have been evolving effect methods of placing information in visual context. There is an art to publishing large amounts of data for easy consumption. Tables, indices, graphs, graphics and typography all combine to make the information easier to consume with the human eye. The World Wide Web took advantage of these advances and changed the Internet from a clunky, intimidating command line interface to the same visual context the consumer uses every day.



the number is a phone number and that the phone number belongs to a person named Michael E. Bailey. There are a lot of inferences here. We are able to make such inference from the metadata provided in the table.

LastName	FirstName	MI	Phone
Ricker	Jeffrey	M	703.555.1234
Bailey	Michael	E	810.555.4321
Scheffer	Linda	C	202.555.2341

• *Figure 2 Sample database table of phone numbers*

The header label *LastName* is metadata about the strings found in the column beneath that header. The header provides information about information.

## HTML AND XML COMPARED

### HTML

We stated earlier that HTML made the Internet one giant library, but XML makes the Internet one giant database. The distinction between a library and a database is an important one.

(Dollars in millions)

	1997	1996
Current assets		
Cash and cash equivalents	\$7,106	\$7,687
Marketable securities	447	450
Notes and accounts receivable	16,850	16,515

*Table 2 Excerpt from a sample balance sheet*

I created the table using a word processor. I could easily save the table in an HTML format so that it could be viewed in a web browser. Listing 1 shows a clean and simple HTML representation of the table. I say “clean and simple” because a more complete HTML representation of the table is easily three pages long. One can be found on the book’s website.

```
<html>
<body>
...
<p><i>(Dollars in millions)</i></p>
<table>
<tr>
  <td>&nbsp;</td>
  <td>1997</td><td>1996</td>
</tr>
<tr>
  <td>Current assets</td>
  <td>&nbsp;</td><td>&nbsp;</td>
</tr>
<tr>
  <td> &nbsp;</td> Cash and cash equivalents</td>
  <td align="right">$ 7,106</td>
  <td align="right">$ 7,687</td>
</tr>
<tr>
  <td> &nbsp;</td> Marketable securities </td>
  <td align="right">447</td>
```

```

<td align="right">450</td>
</tr>
<tr>
<td> &nbsp;     Notes and accounts receivable
</td>
<td align="right">16,850</td>
<td align="right">16,515</td>
</tr>
...
</table>
</body>
</html>

```

*Listing 1 HTML representation of sample balance sheet*

The markup here simply describes how to display the information. You can see the tag *table*. The tag TR indicates a table row and TD indicates a table cell. The purpose of HTML is to place information into visual context. HTML goes to a browser for human consumption, and that's it. HTML is the end of the line for data consumption.

Doing systems integration with HTML is like trying to do systems integration at the printer. It is only human readable at that point. It requires human intervention to make the information machine-readable again. (Of course, many companies are known to do systems integration at the printer.)

The context is the metadata. Visual context simply means that it requires a human eye to extract the metadata.

### XML

Let us continue with our focus on the amount of marketable securities in 1997.

(Dollars in millions)

	1997	1996
Current assets		
Cash and cash equivalents	\$ 7,106	\$ 7,687
Marketable securities	447	450
Notes and accounts receivable	16,850	16,515

We can begin by marking the number 447 as a value.

```
<value>447</value>
```

We can add an attribute to give the information more context. For instance, we could indicate that it is a value for the period 1997.

```
<value period="1997">447</value>
```

We can nest the value tag with another tag to give even more context.

```
<asset type="securities"><value period="1997">447</value></asset>
```

Note that XML requires that tags be properly nested, that is, they cannot overlap. That forces any XML document or fragment into a natural tree structure. A tree structure lends itself nicely to a whole set of mathematics.

We can continue nesting this element, providing more and more context.

```
<balance-sheet>
...
<asset type="current">
  <asset type="securities">
    <value period="1997">447</value>
  </asset>
</total-assets>
...
</balance-sheet>
```

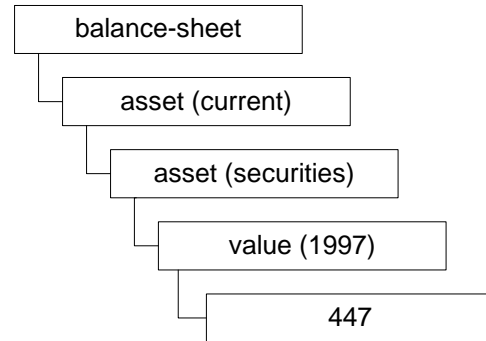


Figure 3 Natural tree structure

The complete example might look like this.

```
<balance-sheet scale="1000000">
<total-assets>
...
<asset type="current">
  <label>Current assets</label>
  <asset type="cash">
    <label>Cash and cash equivalents</label>
    <value period="1997">7106</value>
    <value period="1996">7687</value>
  </asset>
  <asset type="securities">
    <label>Marketable securities</label>
    <value period="1997">447</value>
    <value period="1996">450</value>
  </asset>
  <asset type="receivables">
    <label>Notes and accounts receivable</label>
    <value period="1997">16850</value>
    <value period="1996">16515</value>
  </asset>
  ...
</asset>
</total-assets>
</balance-sheet>
```

Listing 2 XML representation of sample balance sheet

---

## SERIALIZATION

---

Information context is achieved through metadata. Metadata is key to serialization, serialization is key to application integration, and e-commerce is the integration of applications across company boundaries.

Serialization is the process of transforming a complex data structure into a string of text. Serialization can be complex. Take the simple conceptual data structure shown in Figure 4. We can represent the object as a string of text by starting with the parent node A and listing the children B and C. We can then list the grandchildren D and E. The node E is the child of both B and C. We

must not list it twice in the process of serialization because it exists only once. The complexity actually arises when we reconstitute this object. How do we interpret from the string of text that E is the child of B and C but is the same thing?

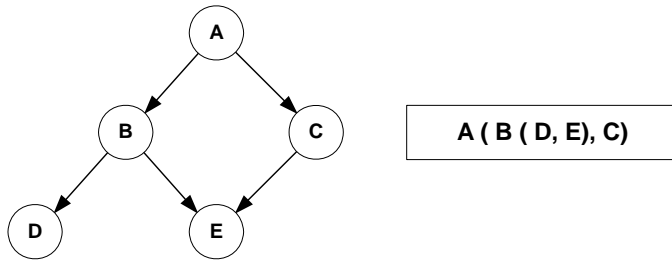


Figure 4 *Serialization is the process of transforming complex objects into a string of text*

XML serves the purpose of serializing complex objects that represent data. Each application has its own internal representation of these objects. XML will not change that fact. The best it can do for internal representations is to effect any new applications that might be built. XML is simply a standard means of serializing a complex internal representation so that it can be sent across the Internet and transformed accurately into another different complex internal representation.

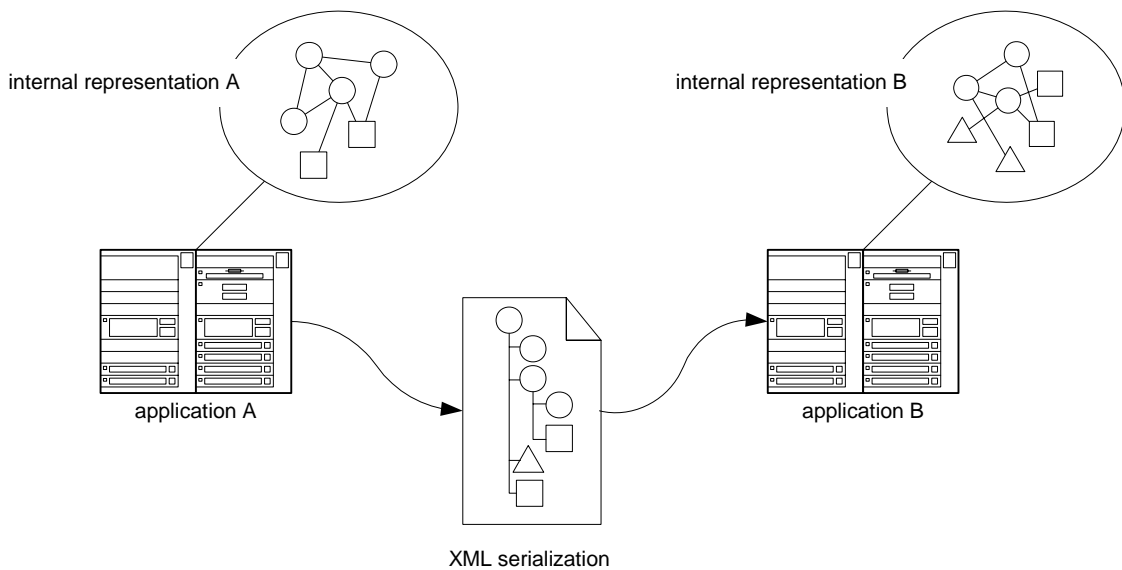


Figure 5 *XML is used to serialize complex objects*

XML is strongly suited for serialization, especially over the Internet. An XML document is a string of text. Its simple rules enable consistent parsing. Its inherent tree structure and built in linking allows XML to represent the most complex data structures. More importantly, it was designed from the ground up for the purpose of exchanging data over the Internet. Most importantly, XML documents are self-describing, that is, they provide the data and the metadata to place information in context.

---

## WEB SERVICES REFERENCE MODEL

---

The objective of electronic commerce is to eliminate manual processes of trade by allowing the internal applications of different companies to exchange information directly. E-commerce is thus a process of integrating applications across corporate boundaries. We call this effort *trans-enterprise integration*, or XEI.

XML is the key technology to making trans-enterprise integration or e-commerce happen. As we have seen, serialization is the key to integration and XML is the standard means for serialization for the Internet.

XML enables a programmer to treat all other data sources as if they were an XML document. Put another way, one can treat applications as if they were documents and documents as if they were applications. With XML, access to a purchase order document has the same power as access to a purchase order application.

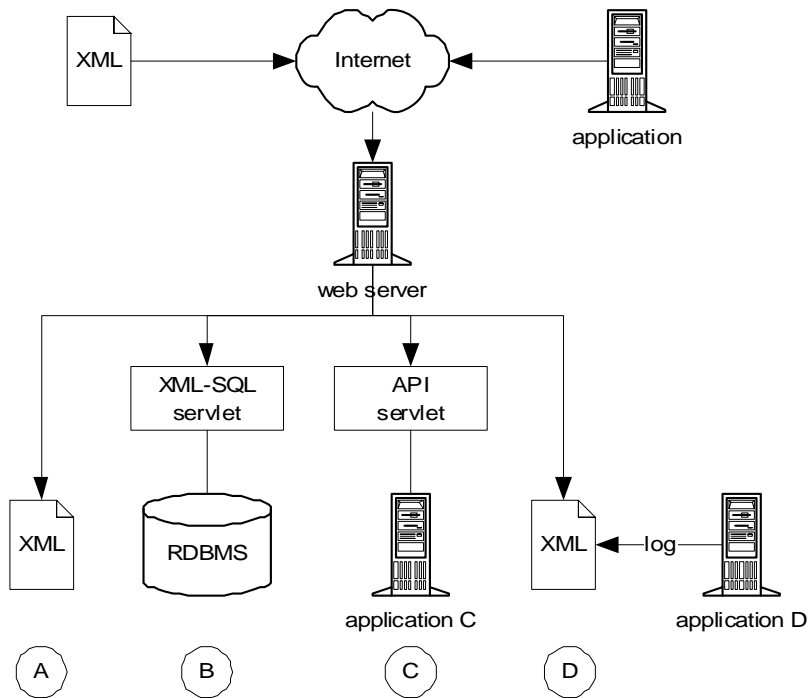


Figure 6 XEI reference model

The basic principle of trans-enterprise integration is that any application or data source can be treated as if it were an XML document accessible over the Internet. We call Figure 6 the web services reference model because it graphically demonstrates this basic principle. Let us briefly describe each component of the reference model.

- **DOCUMENT.** An XML page includes a hyper link to another XML page (A). The URL request is sent to a webservice. The client asks for a XML document, it is sent an XML document. Likewise, an application can ask for an XML document through a URL. Instead of displaying the document, the application is interested in parsing the document and extracting particular values for calculation.

- DATABASE. The client, be it a document or an application, requests a URL that is actually a servlet (B). The servlet translates the URL into an SQL query against a database. The servlet then translates the table result into an XML document. The client asks for an XML document and receives an XML document, but in reality the client completed a database query.
- APPLICATION. The same process works for an application programming interface (API) call. The client requests a XML document from a URL that is a servlet (C). The servlet translates the URL into an API call. The servlet then translates the results into an XML document. The client asks for an XML document and receives an XML document, but in reality the client completed an API call.
- LEGACY APPLICATION. Some applications that do not have an API do have a log of events (D). This log is a direct text representation of the status of the application. By translating the log into an XML document, you can represent the state of the application in a read-only format.

---

### DATABASE EXAMPLE

---

The purpose of this example is to demonstrate the feasibility of treating a relational database as an XML document. Figure 7 shows a very simple relational database table. We will assume that this table is accessible to a Java database connectivity (JDBC) call.

LastName	FirstName	MI	Phone
Ricker	Jeffrey	M	703.555.1234
Bailey	Michael	E	810.555.4321
Scheffer	Linda	C	202.555.2341

• *Figure 7 Sample database table of phone numbers*

The following Java servlet accesses the database with the SQL query

```
SELECT * FROM PEOPLE
```

And translates the results into an XML document. The resulting XML document is shown in Listing 4.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class SimpleDatabaseXML extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

        res.setContentType("text/xml");
        PrintWriter out = res.getWriter();
        out.println("<?xml version='1.0'?">");
        try {
            // need to supply a real JDBC connection URL with real driver
            Connection con =
                DriverManager.getConnection("jdbc:mysql://mydatabase:2356");
            Statement statement = con.createStatement();
```

```

// assuming the table name is PEOPLE
ResultSet rs = statement.executeQuery("SELECT * FROM PEOPLE");
out.println("<people>\n");
while (rs.next()){
    out.println("<person>");
    out.println("<LastName>" + rs.getString("LastName") +
        "</LastName>");
    out.println("<FirstName>" + rs.getString("FirstName") +
        "</FirstName>");
    out.println("<MI>" + rs.getString("MI") + "</MI>");
    out.println("<Phone>" + rs.getString("Phone") + "</Phone>");
    out.println("</person>\n");
}
out.println("</people>\n");
statement.close();
con.close();
}
catch (SQLException e){
    out.println("<error>Trouble opening database.</error>");
}
}
}

```

*Listing 3 Simple JDBC servlet*

```

<?xml version="1.0"?>
<people>
  <person>
    <LastName>Ricker</LastName>
    <FirstName>Jeffrey</FirstName>
    <MI>M</MI>
    <Phone>703.555.1234</Phone>
  </person>
  <person>
    <LastName>Bailey</LastName>
    <FirstName>Michael</FirstName>
    <MI>E</MI>
    <Phone>810.555.4321</Phone>
  </person>
  <person>
    <LastName>Scheffer</LastName>
    <FirstName>Linda</FirstName>
    <MI>C</MI>
    <Phone>202.555.2341</Phone>
  </person>
</people>

```

*Listing 4 Resulting XML from the JDBC servlet*

Application integration would be very much like the example we have just walked through except, instead of making an SQL query, the code would make a proprietary API call. The code would then need to translate the API results into some XML form.

---

### SIMPLE OBJECT ACCESS PROTOCOL (SOAP)

---

Simple Object Access Protocol (SOAP) provides a basis for integrating web services. It is an exceedingly simple protocol that provides the lowest common denominator for XML-based application integration. In essence, SOAP dictates that an object such as our database servlet above should accept its parameters and publish its results using XML. It further dictates that the XML of

these messages should be wrapped in an envelop with a header and a body. The following listing shows the XML wrapped in a SOAP envelope.

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Header/>
  <env:Body>
    <people>
      <person>
        <LastName>Ricker</LastName>
        <FirstName>Jeffrey</FirstName>
        <MI>M</MI>
        <Phone>703.555.1234</Phone>
      </person>
      <person>
        <LastName>Bailey</LastName>
        <FirstName>Michael</FirstName>
        <MI>E</MI>
        <Phone>810.555.4321</Phone>
      </person>
      <person>
        <LastName>Scheffer</LastName>
        <FirstName>Linda</FirstName>
        <MI>C</MI>
        <Phone>202.555.2341</Phone>
      </person>
    </people>
  </env:Body>
</env:Envelope>
```

*Listing 5 Sample results as SOAP message*

---

## SUCCESS THROUGH SIMPLICITY

---

Like HTML before, the success of XML demands that the most basic e-commerce solution must be free and simple. That means simple, rapidly fielded open source e-commerce standards.

HTML exploded because the basic software necessary was free and the basic knowledge necessary was simple. Most people could download the software, learn the basics and have a website up in a day. The basic requirements of creating a website are still free today, and yet companies pay thousands of dollars for web servers and hundreds of thousands for websites.

Most e-commerce software vendors have built business models that demand that even the most humble user pay a high price to use e-commerce. With competing XML-based e-commerce solutions already emerging, the market falls under a corollary to Moore's Law that I call Munro's Law:

*The value of an e-commerce standard is exponentially proportional to the number of companies that use that standard.*

A small company's website consists of a set of flat files containing HTML code and content. A large company's website consists of large, complex applications that dynamically generate HTML pages from content stored in relational databases. Nevertheless, to the client they are both HTML pages.

E-commerce must work the same way. Large companies will generate e-commerce XML through applications linked to their enterprise resource planning (ERP) systems, etc. The small companies will simply post flat files of XML to a web directory. To the e-commerce client, they will both be the same. That is why the Simple Object Access Protocol begins with the word *simple*. Efforts to add complexity to web services and e-commerce are misguided and self-defeating. These efforts begin with a misguided understanding on the part of some engineers on how technologists add value to their organizations.

---

### VALUE OF THE TECHNOLOGIST

---

Communication technology can be rather intimidating at first. Legacy electronic commerce systems have their cryptic message formats and their private networks. Little if any documentation is readily available. It's hard to even find an example. The whole thing takes on the air of a mysterious priesthood whose secrets we could not possibly hope to recover.

There is a group of people who actually foster and perpetuate this feeling of unfathomable mystery. Let's face it; there are a rather large number of technology professionals who earnestly believe that their value rests in making things complicated. The more mucked up the technology is, the fewer people that understand it, the more important they are because they understand the mucking. The customer, client or the employer is to feel ingratiated to them for saving everyone from the muck. More and more customers and employers are realizing that those whom they are suppose to thank (and pay) for saving them from a problem are the same people who created the problem in the first place. Feelings move quickly from gratitude right past resentment to loathing.

The Year 2000 bug did much to bring attention to this sort of behavior.

The true value of a technologist lies in making things simpler, cheaper, easier and more understandable. Sir Isaac Newton didn't start a cult of secrecy when he discovered his three laws of mechanics. He published them as quickly as possible in a language most readily accessible to anyone living in the 17<sup>th</sup> century. So it has been since his time (or rather Hume's time).

---

### INVERTED PYRAMID

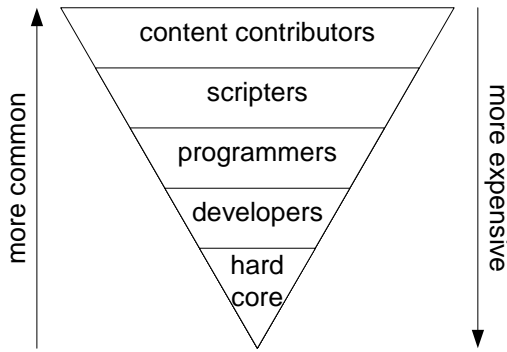
---

The information technology talent pool can be thought of as an inverted pyramid. Most any one that works in an office today is a content contributor. They use word processors, spread sheets and other graphical user interfaces (GUI) to generate content in a machine-readable format.

Just below content contributors are scripters. Scripters can write HTML and XML. Many can write Java scripts or Visual Basic scripts, macros, even Perl scripts.

Programmers can write in programming languages such as C, Java, Visual Basic and Perl. Their programs however are intended for an office-level environment and lack key concepts such as multithreading, load balancing, exception handling, etc. Developers, on the other hand, understand these other advanced concepts. Developers create shrink-wrapped software.

At the bottom of the pyramid are hard-core developers. These people create the infrastructure that every one else assumes, such as operating systems, compilers and routers.



*Figure 8 The technology talent pool as an inverted pyramid*

The further down the pyramid one goes, the harder that talent is to find. The simple rules of supply and demand make hard-core developers very expensive.

The value of a technologist does not lie in making technology more complicated. The true value of a technologist lies in enabling the level above. The value of a developer lies in how well he or she eases the burden on the programmer and scripter. The value of the programmer lies in how well he or she eases the burden of the scripter and the contributor, and so on.

The implication of XML in this pyramid concept is very important: XML ALLOWS SYSTEMS INTEGRATION TO OCCUR AT THE SCRIPTER LEVEL. The key breakthrough in XML is not so much the XML data format as it is the XML parser. Any scripter can now publish data that is machine-readable. More importantly, now any scripter has access to the machine that actually does read the data. The free and widely distributed XML parsers are what make XML work. The hard core developers made that possible.