

Ten XML Schemas you should know

Solutions for sharing data between different platforms and environments

Skill Level: Introductory

[Martin Brown](#)

Developer and writer
Freelance

01 Jul 2008

In this article, look at some top XML schemas that provide solutions for all sorts of problems, from the basics of Web services to data description. You'll also cover database-like solutions that involve contacts and invoices. The schemas in this article were chosen for their usefulness and utility, plus their impact on the XML community in how information is shared and exchanged using the XML format.

SOAP

Frequently used acronyms

- Ajax: Asynchronous JavaScript + XML
- CPU: Central Processing Unit
- CSS: Cascading Stylesheets
- HTML: Hypertext Markup Language
- OASIS: Organization for the Advancement of Structured Information Standards.
- PDF: Portable Document Format
- W3C: World Wide Web Consortium
- XHTML: Extensible HyperText Markup Language
- XML: Extensible Markup Language

- XSLT: Extensible Stylesheet Language Transformations

The Simple Object Access Protocol (SOAP) is actually a Web services technology, but the format of data exchange between the client and the server in the Web service is through a flexible XML schema.

One main benefit of Web services is the level of interoperability of the information and data exchanged between the client and the server over the network. The SOAP standard uses XML to help structure the data and to help define the data types and information in an architecture neutral format.

From your chosen programming language, you just supply the data type and the name of the function to be called on the remote server. The SOAP library converts the information from the host language and format into an XML-formatted message that contains the function called and the arguments supplied.

You can see the structure of SOAP if you look at the W3 Consortium's own examples. When calling the remote SOAP function `GetEndorsingBoarder()`, the client caller generates an XML message like the one in Listing 1.

Listing 1. Calling the remote SOAP function `GetEndorsingBoarder()`

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" >
  <SOAP-ENV:Body>
    <m:GetEndorsingBoarder xmlns:m="http://namespaces.snowboard-info.com">
      <manufacturer>K2</manufacturer>
      <model>Fatbob</model>
    </m:GetEndorsingBoarder>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The entire message sent by the SOAP client is encoded into what is known as the SOAP envelope. The contents of the envelope form the detailed content of the message.

The called method is clearly identified as `GetEndorsingBoarder`, and it accepts two arguments, the manufacturer and the model. You can see here how the translation from a native, possibly binary-based format for a string was turned into an XML string. Because XML is platform independent, hosts can swap messages using the SOAP system without complex binary encoding or decoding.

The response from the server is sent back as another XML-encoded, SOAP envelope, this time with the return value from the function. Responses from a SOAP request are always formatted as the same function, with `Response` appended to the end of the envelope detail, as you can see in Listing 2.

Listing 2. Response from a SOAP request

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetEndorsingBoarderResponse xmlns:m="http://namespaces.snowboard-info.com">
      <endorsingBoarder>Chris Englesmann</endorsingBoarder>
    </m:GetEndorsingBoarderResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Typically, you don't write these SOAP messages yourself; the SOAP library handles that for you. But the structure and simplicity of the SOAP envelope demonstrates how easy it is to share information using the SOAP standard.

SOAP has significantly simplified how you can exchange messages and call remote functions. The Remote Procedure Call (RPC) standard requires complex methods to handle serialization of the binary data, and to send more structured information requires a detailed declaration and conversion of the source material each way.

With SOAP, the serialization into XML eliminates a lot of this complexity, and makes cross-platform and cross-language integration, and data exchange much easier.

WSDL

The Web Services Description Language (WSDL) provides a convenient method to describe Web services (mostly using SOAP). WSDL enables you to describe the services and interfaces made available using the SOAP standard.

For example, you can create a WSDL file that describes the services available on an individual server and then distributes this file to Web service consumers that need to use these services. By reading and parsing the WSDL file, the consumers can learn everything they need to know about how to use the Web services, including the data types and arguments that can be exchanged, and the different errors and other information that is returned.

Using the example from the W3 Consortium again, you can see that the declaration of the different remote functions and the data to be exchanged is all handled through the XML definition of the structures, as in Listing 3.

Listing 3. The XML definition the different remote functions and the data to be exchanged

```
<?xml version="1.0"?>
<!-- root element wsdl:definitions defines set of related services -->
<wsdl:definitions name="EndorsementSearch"
```

```

targetNamespace="http://namespaces.snowboard-info.com"
xmlns:es="http://www.snowboard-info.com/EndorsementSearch.wsdl"
xmlns:esxsd="http://schemas.snowboard-info.com/EndorsementSearch.xsd"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

<!-- wsdl:types encapsulates schema definitions of communication types;
      here using xsd -->
<wsdl:types>

  <!-- all type declarations are in a chunk of xsd -->
  <xsd:schema targetNamespace="http://namespaces.snowboard-info.com"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">

    <!-- xsd definition: GetEndorsingBoarder [manufacturer string,
      model string] -->
    <xsd:element name="GetEndorsingBoarder">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="manufacturer" type="string"/>
          <xsd:element name="model" type="string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <!-- xsd definition: GetEndorsingBoarderResponse
  [... endorsingBoarder string ...] -->
    <xsd:element name="GetEndorsingBoarderResponse">
      <xsd:complexType>
        <xsd:all>
          <xsd:element name="endorsingBoarder" type="string"/>
        </xsd:all>
      </xsd:complexType>
    </xsd:element>

    <!-- xsd definition: GetEndorsingBoarderFault
  [... errorMessage string ...] -->
    <xsd:element name="GetEndorsingBoarderFault">
      <xsd:complexType>
        <xsd:all>
          <xsd:element name="errorMessage" type="string"/>
        </xsd:all>
      </xsd:complexType>
    </xsd:element>

  </xsd:schema>
</wsdl:types>

<!-- wsdl:message elements describe potential transactions -->

<!-- request GetEndorsingBoarderRequest is of type GetEndorsingBoarder -->
<wsdl:message name="GetEndorsingBoarderRequest">
  <wsdl:part name="body" element="esxsd:GetEndorsingBoarder"/>
</wsdl:message>

<!-- response GetEndorsingBoarderResponse is of type
      GetEndorsingBoarderResponse -->
<wsdl:message name="GetEndorsingBoarderResponse">
  <wsdl:part name="body" element="esxsd:GetEndorsingBoarderResponse"/>
</wsdl:message>

<!-- wsdl:portType describes messages in an operation -->
<wsdl:portType name="GetEndorsingBoarderPortType">

  <!-- the value of wsdl:operation eludes me -->
  <wsdl:operation name="GetEndorsingBoarder">
    <wsdl:input message="es:GetEndorsingBoarderRequest"/>
    <wsdl:output message="es:GetEndorsingBoarderResponse"/>
    <wsdl:fault message="es:GetEndorsingBoarderFault"/>
  </wsdl:operation>
</wsdl:portType>

```

```

    </wsdl:operation>
</wsdl:portType>

<!-- wsdl:binding states a serialization protocol for this service -->
<wsdl:binding name="EndorsementSearchSoapBinding"
              type="es:GetEndorsingBoarderPortType">

  <!-- leverage off soap:binding document style ...(no wsdl:foo pointing at
the soap binding) -->
  <soap:binding style="document"
                transport="http://schemas.xmlsoap.org/soap/http"/>

  <!-- semi-opaque container of network transport details classed by
soap:binding above ... -->
  <wsdl:operation name="GetEndorsingBoarder">

    <!-- again bind to SOAP? ... -->
    <soap:operation soapAction="http://www.snowboard-info.com/
EndorsementSearch"/>

    <!-- further specify that the messages in the wsdl:operation
"GetEndorsingBoarder" use SOAP? ... -->
    <wsdl:input>
      <soap:body use="literal"
                namespace="http://schemas.snowboard-info.com/EndorsementSearch.xsd"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"
                namespace="http://schemas.snowboard-info.com/EndorsementSearch.xsd"/>
    </wsdl:output>
    <wsdl:fault>
      <soap:body use="literal"
                namespace="http://schemas.snowboard-info.com/EndorsementSearch.xsd"/>
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>

<!-- wsdl:service names a new service "EndorsementSearchService" -->
<wsdl:service name="EndorsementSearchService">
  <wsdl:documentation>snowboarding-info.com Endorsement Service</
wsdl:documentation>

  <!-- connect it to the binding "EndorsementSearchSoapBinding" above -->
  <wsdl:port name="GetEndorsingBoarderPort"
              binding="es:EndorsementSearchSoapBinding">

    <!-- give the binding an network address -->
    <soap:address location="http://www.snowboard-info.com/EndorsementSearch"/>
  </wsdl:port>
</wsdl:service>

</wsdl:definitions>

```

The WSDL declares the message types, the fault types and contents, and the structure of the data to be exchanged.

Everything you need to use and access the SOAP interface to the server is available right here in the WSDL. Most languages and environments have a mechanism to read and parse the WSDL to determine the functions and data exchange that is available.

WSDL not only defines the SOAP interfaces that you can use to exchange

information, with suitable WSDL generators, it can also be used to create the code required to send the request, plus generate and format the response.

WSDL and SOAP combined together make a powerful remote procedure calling system.

RDF

The Semantic Web and Semantic Grid technologies both rely on a flexible description language offered by the Resource Description Framework (RDF). The RDF format is really a family of standards. It enables you to describe information and resources in a way that allows systems to easily make connections and relations between different resources.

RDF is another W3C-approved standard, and is a standard for the definition of the information and resources. RDF doesn't require XML, but one of the serialization formats used to describe the information uses XML.

To define a resource, you specify a subject-predicate-object expression. For example, when you describe the content of a Web site, the subject is the Web site, the predicate is "contains information about" and the object would be the content type. To create a relationship between that Web site and another resource, you use the Friend of a Friend (FOAF) notation to create the link between the two resources.

The purpose of RDF is to map what are natural language statements about resources and information into a format that a computer can parse. For example, you might rewrite the statement: *The MCSLP.com Website is authored by Martin C Brown* into the RDF XML in Listing 4.

Listing 4. Rewriting a statement in RDF XML

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:si="http://www.recshop.fake/siteinfo#">
  <rdf:Description rdf:about="http://www.mcslp.com/">
    <si:author>Martin C Brown</si:author>
  </rdf:Description>
</rdf:RDF>
```

Another example of the utility of the RDF standard is that the early syndication systems provided by news sites and blogs used the RDF specification to define the feed content and individual stories. You can see an example of this in Listing 5.

Listing 5. Using the RDF specification to define the feed content and individual stories

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns="http://my.netscape.com/rdf/simple/0.9/">

  <channel>
    <title>MCslp</title>
    <link>http://www.mcslp.com</link>
    <description>MCslp Projects</description>
  </channel>

  <item>
    <title>Voice enabling XML</title>
    <link>http://mcslp.com/?p=295link</link>
  </item>

  ...

</rdf:RDF>
```

Originally the RDF standard was designed for the specific purpose of describing resources, content, and relationships on the Web. However, the RDF standard is now used as the standard to describe general information, resources, and relationships.

The Semantic Web and Grid technologies both rely on the definition of resources and relationships to enable applications to make use of the different pieces of information and bond that data together.

vCard

Recording contact information is a vital part of nearly all business applications, and the capture of that information in an effective XML structure can make working with the data much easier.

Using XML is an obvious choice for contact information because the great variability in the information. For example, some companies and individuals will have multiple addresses, telephone numbers, and e-mail accounts. To declare multiple fragments of information like this becomes easy to achieve within the XML structure.

The vCard structure is frequently used on the Internet to represent contact information in a platform-independent way that is easy to generate and import into different applications. It supports some of the flexibility of the XML structure, but is, in fact, a simple text-based format that uses declared fields and extensions to provide the information. Unlike XML, the vCard format is flat file, which means that you cannot directly attach information to different elements. A good example here is a telephone number, which cannot be associated with an explicit address, only identified as another phone number for the record.

The W3 Consortium suggests an XML representation of the vCard format that is

actually based on the RDF XML standard and allows you to more easily format and exchange the contact information. Using RDF as a framework enables you to retain some of the structural information during the declaration. For example, the RDF standard supports the use of the bags, sequences, and alternatives during data description. Bags support multiple declarations for an object, such as multiple roles, and can be used when the sequence is not important. Sequences support a defined sequence for an object, such as the hierarchy of a person's role within in an organization. Alternatives allow you to choose one item from a list, such as multiple e-mail addresses.

Listing 6 shows a typical vCard for the fictional Charles Perston.

Listing 6. Typical vCard for the fictional Charles Perston

```
BEGIN:VCARD
VERSION:3.0
N:Perston;Charles;;;
FN:Charles Perston
ORG:Perston Technology;
EMAIL;type=INTERNET;type=WORK;type=pref:null@perston.co.uk
TEL;type=WORK;type=pref:01234 567890
item1.ADR;type=WORK;type=pref;;;Perston House;Perston;Perstonshire;P1 0NS;UK
item1.X-ABADR:gb
X-ABUID:5AE47BB6-4E0F-4558-980C-BD3066FA6154\ :ABPerson
END:VCARD
```

Using the vCard XML standard, you can represent the same information using the structure in Listing 7.

Listing 7. Using the vCard XML standard to represent the fictional Charles Perston

```
<vCard:vCard xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/" vCard:version="3.0"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" vCard:class="PUBLIC"
  xmlns:vCard="x-urn:cpan:ascope:xml-generator-vcard#">
  <vCard:fn>Charles Perston</vCard:fn>
  <vCard:n>
    <vCard:family>Perston</vCard:family>
    <vCard:given>Charles</vCard:given>
  </vCard:n>
  <vCard:adr vCard:del.type="pref;work">
    <vCard:street>Perston House</vCard:street>
    <vCard:locality>Perston</vCard:locality>
    <vCard:region>Perstonshire</vCard:region>
    <vCard:pcode>P1 0NS</vCard:pcode>
    <vCard:country>UK</vCard:country>
  </vCard:adr>
  <vCard:email vCard:email.type="internet;pref;work">null@perston.co.uk
  </vCard:email>

  <vCard:org>
    <vCard:orgnam>Perston Technology</vCard:orgnam>
  </vCard:org>
</vCard:vCard>
```

The XML format is more verbose, but it is easy to understand what you are looking at, and how the relationships between the different components work together. You also get more specific information and detail in the format. For example, you can easily identify the country in the address, data that was comparatively hidden in the standard vCard output.

For example, you might easily, use XPath or SAX events to pull out a list of the countries to learn the number of contacts in different places.

DocBook XML

The ability to write a document and then create it in a variety of different output formats was a dream for many organizations for many years. With DocBook XML you can achieve that, and still keep both the semantic markup as well as retain control over the formatting and output of the material.

For the semantic control, you create a document by specifying the different chapters, sections, and paragraphs that make up the document. Within the paragraph, you can be more specific about the items that you include. For example, you can encapsulate commands and function names within their own tags, as in Listing 8.

Listing 8. Encapsulating commands and functions in their own tags

```
<para>The <command>compile</command> takes the source code of the
material and builds a new class based on the filename. For example, if the filename is
<filename>HelloWorld</filename> then the name of the class generated will be
<classname>HelloWorld</classname>.
```

When the individual elements are generated you can choose a different output style and format for each one, or use the same one. More critically, because the semantic information is returned (for example, the documentation might contain a reference to a class name), you can use that during indexing to produce a list of all the class names detailed in the document.

In addition to this semantic markup, the individual sections and components of the document can be marked up with specific IDs, and you can use these IDs to provide links between different areas of the documentation. For certain types, this is done automatically (sections, chapters, parts, and other types, which ultimately generate the table of contents), whereas for others you can make explicit links to other sections.

When converted into a target format, these links are automatically translated into a format appropriate for the target. When you produce HTML, for example, the link becomes a link to the appropriate HTML page or anchor within the page. When you generate a PDF, you can include the page number of the target section.

This type of translation is handled by the XSLT stylesheets. The standard DocBook XSLT stylesheets available for XML can already convert information into standard HTML, XHTML, PDF (through the FO standard), Texinfo, Java™ Help and Man pages. You can also use the standard stylesheets to output the source material in a variety of sizes and styles, from books, to A4 and even slides.

The flexibility of all of the different output formats and markup means that when you create documentation, you can use the same documentation source to provide a printed manual, your embedded help, man pages, and online and context sensitive information. Using more traditional models, you might develop each of these elements individually.

DocBook XML has gained widespread acceptance in the technical writing community, with a number of different companies using the DocBook XML standard (or subsets of it) to produce all of their documentation.

FIXML

FIX is one of many business-to-business based exchange formats that allow financial institutions to exchange information. A lot of the exchange of this information is fairly significant, from the transfer of data during a transaction of money from one account to another, to the simple exchange of stock prices and trade information.

This is all information that requires transfer, sometimes in relatively small discrete packets, and sometimes in much larger blocks of data. The traditional format for the exchange of this information was a simple key/value pairs for the data, but the sizes of the information exchanged turned out to be inefficient. By using XML they were able to simplify the structure, particularly for complex data.

With an optimized version of the FIXML XML-based version of the data exchange, the developers managed to reduce the size of the data files considerably, and simultaneously make the files more human-readable. Sizes for the exchange of stock data were reduced to almost a quarter of the size when using the old format.

Outside of a typical financial institution, FIXML has little relevance. But if using FIXML enables more efficient trades of information then everybody, ultimately, stands to benefit.

SVG

Scalable Vector Graphics (SVG) is an XML standard for describing illustrations. With SVG, you can describe the lines, shapes, and locations and relations between those elements. Most attractive is that the information can then be output in your desired

format, including a scalable diagram or as a fixed graphic.

SVG fixes a number of the serious problems in the traditional production process for illustrations. Typically these are developed using a specialized illustration program. Sharing information and illustrations between different programs is often very difficult. Saving the file to SVG means that any SVG-capable application can read and work with the file.

The other problem with illustrations is that when they are output in most formats (particularly the Web), the illustrations need to be rendered into a bitmap-based format (such as JPEG, or PNG) before you can display or incorporate them into another document. This traditional approach has a number of problems. First, changes to the original illustration have to be explicitly (and often manually) exported into the bitmap format.

Second, because the bitmap format is based on a pixel-by-pixel representation of the original drawing, you need to select the size and resolution of the generated file carefully to match the output target to ensure the quality of the generated image. For example, the on-screen resolution needs to be 72dpi (or 96dpi) to match the standard resolution of most monitors. For printed output, the bitmap version needs to be from 300 to 2400 DPI. As a consequence, generating the image can also generate very large files in comparison to the source description.

Although vector-based formats existed before PostScript and Encapsulated PostScript, they were very CPU intensive, which made them inefficient for on-screen representations.

With SVG, as with any vector graphics format, the description of the image content is based on listing the different shapes, rather than producing a pixel-by-pixel representation. For example, to describe a rectangle, you specify the starting point for the top left corner, and then the length of the two sides. The image description is represented in XML. Specific tags are available to describe lines, rectangles, polygons, and circles, among others, and you have full control over the style and formatting of the different elements.

You can see an example of this in Listing 9. Here a simple square is drawn, with a transparent circle and triangle.

Listing 9. Drawing a simple square with transparent circle and triangle

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">

<polygon
points="200,100 300,200 150,250"
```

```
style="fill:#cccccc;
stroke:#000000;stroke-width:1"/>

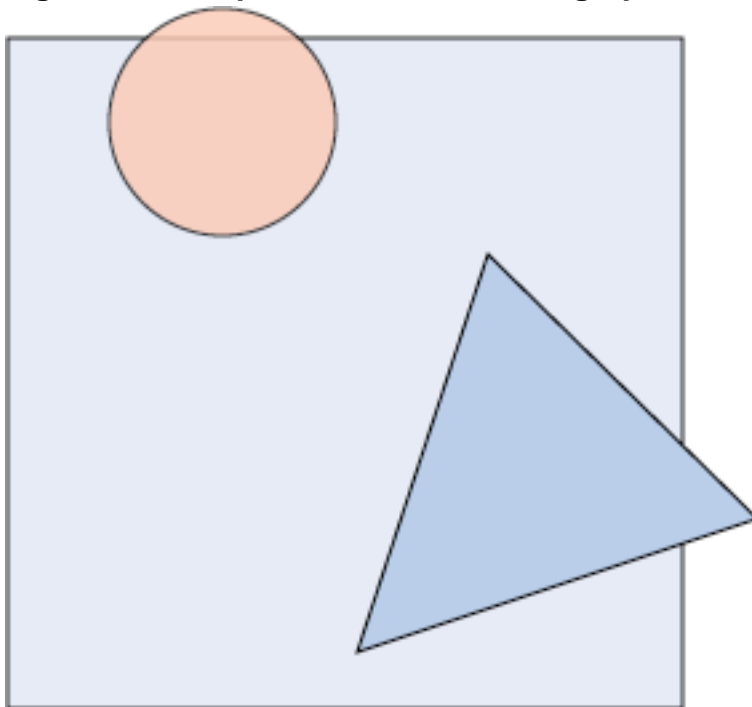
<rect x="20" y="20" width="250" height="250"
style="fill:blue;stroke:black;stroke-width:1;
fill-opacity:0.1;stroke-opacity:0.9"/>

<circle cx="100" cy="50" r="40" stroke="red"
fill="red" style="fill-opacity:0.5"/>

</svg>
```

Figure 1 shows a bitmap version of the rendered graphic.

Figure 1. Bitmap version of rendered graphic



As an SVG file, the description of the image is just over 500 bytes, and as a PNG is almost 9KB.

Using SVG makes your illustrations much smaller, easier to use, and more compatible with a wide range of applications.

Dublin Core

The Dublin Core standard is a method to classify information most commonly used in libraries. A XML Schema exists that works with the Dublin Core standard to define such information using XML. You can use Dublin Core to catalog all sorts of information in an effective manner that is both easy to update and easy to query and use.

The current use of Dublin Core within the description and definition of information is helping the Semantic Web become a reality. By having a universal standard to describe data, and more importantly, using a well-established and proven solution, you can provide a detailed description of the data in another XML document and have the information exchanged and compared effectively between different sources.

The Dublin Core specification has its own schema, but it is designed to be part of a larger XML document, with an XML namespace to define the DC elements required to describe the data in the rest of the document. To see an example of this, look at how the DC classification system is used within the RDF XML schema to describe the content of an RDF entity, such as a Web site (see Listing 10). To do this, you can extend the structure used in an earlier example from your look at the [RDF schemas](#).

Listing 10. The DC classification system used within the RDF XML schema to describe the content of an RDF entity

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://my.netscape.com/rdf/simple/0.9/"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rss:channel rdf:about="http://www.xml.com/xml/news.rss">
    <rss:title>MCSLP</rss:title>
    <rss:link>http://mcslp.com/rss </rss:link>
    <dc:description>
      MCSLP features information, projects and articles from members of the MCSLP team.
    </dc:description>
    <dc:subject>MCSLP, Grids, XML, Databases, Programming </dc:subject>
    <dc:identifier>http://www.mcslp.com</dc:identifier>
    <dc:publisher>MCSLP</dc:publisher>
    <dc:rights>Copyright 2008, MCSLP</dc:rights>
  </rss:channel>
</rdf:RDF>
```

In [Listing 10](#), you used the DC elements to add description, subject, publisher, rights, and identifier information to help classify the contents of the RSS feed.

The full Dublin Core Metadata Elements Set consists of 15 metadata elements:

1. Title
2. Creator
3. Subject
4. Description
5. Publisher

6. Contributor
7. Date
8. Type
9. Format
10. Identifier
11. Source
12. Language
13. Relation
14. Coverage
15. Rights

This provides ample scope to help describe information.

XForms

The XForms XML standard allows you to define the different components of a form (fields, entry types such as radio buttons, or lists) and the validation for the information that you want to be supplied within the form.

The XForms XML standard is very similar to the HTML and XHTML form markup familiar to many Web developers already, and will be incorporated into the XHTML 2.0 standard.

XForms XML is based on simple Model, View, Controller format, where the model is the overall description of the form, its fields, input constraints, and the mode of the submission of that data. The view defines the controls that appear on the form, their grouping, and the fields in the model that they refer to. Formatting and rendering of the individual controls on the form is handled with CSS.

The XForms standard extends the typical HTML form definition through more specific classification of the information to be supplied on the form. You can use dynamic elements (currently normally only supported through the use of JavaScript or Ajax elements) while the form is populated.

You can see an example of a simple text entry and pop-up selection box in Listing 11.

Listing 11. Simple text entry and pop-up selection box

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:xforms="http://www.w3.org/2002/xforms">
  <head>
    <title>XForms Sample</title>
    <xforms:model>
  <xforms:instance>
    <Name xmlns="">
      <FName />
      <LName />
      <Title />
    </Name>
  </xforms:instance>
  </xforms:model>
  </head>
  <body>
  <xforms:select1 ref="Title">
  <xforms:label>Title:</xforms:label>
  <xforms:item>
    <xforms:label>Mr</xforms:label>
    <xforms:value>Mr</xforms:value>
  </xforms:item>
  <xforms:item>
    <xforms:label>Mrs</xforms:label>
    <xforms:value>Mrs</xforms:value>
  </xforms:item>
  </xforms:select1>
  <xforms:input ref="FName">
    <xforms:label>First name: </xforms:label>
  </xforms:input>
  <xforms:input ref="LName">
    <xforms:label>Last name: </xforms:label>
  </xforms:input>
  <hr />
  <xforms:output value="concat('Hello ',Title,' ',FName,' ',LName)"/>
    <xforms:label>Output: </xforms:label>
  </xforms:output>
  </body>
</html>
```

The Firefox XForms extension can be used to view the XForms form in HTML. You can see a sample of this in Figure 2.

Figure 2. Using the Firefox XForms extension to view the XForms form in HTML

Title: First name: Last name:

Output: Hello Mr Martin Brown

Customer invoicing

An age-old problem for many business processes is migrating the paper-based structure of a customer invoicing system into a computer process. To create an invoice structure requires careful consideration of the various types and repeating

elements to make a workable structure.

In the past, the exchange of business information like invoices has been handled in the past by creating very large structures and definitions; the international standard for exchanging invoice information contains hundreds of possible fields of information. Without an effective method to exchange that data, share invoicing, order, and other data can be difficult.

Without a universally accepted standard, many organizations have developed and recommended different versions of the core invoice standard. Of those available, the one developed by the OASIS group is probably the best known, and is the one agreed upon by a large number of companies and organizations.

The structure is part of a larger framework that OASIS has developed called the Universal Business Logic (UBL) and it includes both the schemas and the workflows for everything from the order processing through to the invoicing and payment. The complexity of the system is too much to go into as part of this article, but if you want a flexible, interoperable system, then UBL is a good place to start.

Summary

In this article, you looked at a wide range of different XML schemas that provide functionality and structure for everything from simple description frameworks (RDF), to graphics formats (SVG), and even entire structures for business workflow (UBL). In each case, you saw how the flexibility of the structure and content of the XML standard make the development of each system so easy. In addition, the cross-platform compatibility of XML makes it an ideal choice where you have to share data between different platforms and environments. For WSDL and SOAP, it is one of the most important features.

Resources

Learn

- [XML.org](#): Visit this site to see information on a wide range of XML schemas and standards, including SOAP, WSDL and SVG.
- [The OASIS Universal Business Logic \(UBL\)](#): Learn about a complete order processing system and workflow using XML.
- [The XForms standard](#): Learn about one of the XML standards handled by the W3 Consortium.
- [The Dublin Core Metadata Initiative \(DCMI\)](#): Visit the home page of the organization responsible for developing and specifying the Dublin Core system to describe information within XML documents.
- [Scalable Vector Graphics \(SVG\)](#): Get information on this standard to describe graphics in a vector format.
- [vCard XML](#): Learn about the standard to represent contact information in an interoperable way using XML.
- [Introduction to XML](#) (Doug Tidwell, developerWorks, August 2002): Get a basic grounding in XML from this classic tutorial.
- [Using WSDL in SOAP applications](#) (Uche Ogbuji, developerWorks, November 2000): Read a solid introduction to WSDL for SOAP programmers.
- [Introduction to Scalable Vector Graphics](#) (Nicholas Chase, developerWorks, March 2004): Learn about the concepts necessary to build SVG documents, such as basic shapes, paths, text, and painting models, plus animation and scripting.
- [Understanding XForms](#) (Kurt Cagle, developerWorks, June 2007): Explore more background on XForms.
- [An introduction to RDF](#) (Uche Ogbuji, developerWorks, December 2000): Learn about the origins of RDF as well as RDF aspects from schemas to usage scenarios.
- [SVG and XForms: A primer](#) (Antoine Quint, developerWorks, November 2003): Get an overview of the two technologies that highlights the potential synergies between them.
- [IBM XML certification](#): Find out how you can become an IBM-Certified Developer in XML and related technologies.
- [XML technical library](#): See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.

- [developerWorks technical events and webcasts](#): Stay current with technology in these sessions.
- The [technology bookstore](#): Browse for books on these and other technical topics.
- [developerWorks podcasts](#): Listen to interesting interviews and discussions for software developers.

Get products and technologies

- [IBM trial software for product evaluation](#): Build your next project with trial software available for download directly from developerWorks, including application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- [XML zone discussion forums](#): Participate in any of several XML-related discussions.
- [developerWorks XML zone: Share your thoughts](#): After you read this article, post your comments and thoughts in this forum. The XML zone editors moderate the forum and welcome your input.
- [developerWorks blogs](#): Check out these blogs and get involved in the [developerWorks community](#).

About the author

Martin Brown

Martin Brown has been a professional writer for over eight years. He is the author of numerous books and articles across a range of topics. His expertise spans myriad development languages and platforms -- Perl, Python, Java, JavaScript, Basic, Pascal, Modula-2, C, C++, Rebol, Gawk, Shellscript, Windows, Solaris, Linux, BeOS, Mac OS/X and more -- as well as Web programming, systems management and integration. Martin is a regular contributor to ServerWatch.com, LinuxToday.com and IBM developerWorks, and a regular blogger at Computerworld, The Apple Blog and other sites, as well as a Subject Matter Expert (SME) for Microsoft. He can be contacted through his Web site at <http://www.mcslp.com>.

Trademarks

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered

trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli, WebSphere, and pureXML are trademarks of IBM Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.