Back to the Future with XML Schema

Dennis Remmer
E2, Australia

Summary

A general overview is presented of the XML Schema specification ratified by the World Wide Web Consortium (W3C) as part of its framework of XML standards. XML Schema provides the long-awaited capability to properly express classes of information for XML-consuming applications.

Previously, Document Type Definitions (DTDs) gave XML information and application designers a means for modelling information, but severe limitations with the standard meant that database professionals (in particular) were unable to fully embrace XML to meet their needs for distributed information interchange. The XML Schema specification is explored, along with its evolution, purpose, and usage. Oracle's support for the specification in its XML Developers Kit (XDK) and 9i platform is examined, along with alternatives, issues, and available tools.

Introduction

The World Wide Web Consortium (W3C) is a standards development body which is responsible for the definition of XML and its associated specifications. XML is associated with other related W3C standards that, when combined, give us a *framework* for technologies that can service many of the needs of knowledge management. E-Business requires the ability to define and exchange data as seamlessly as possible between applications and services. XML can address such requirements:

- as a way of structuring information
- as a way of setting a common vocabulary
- as a language for formatting data
- by supporting the concept of a ‘self-describing’ document
- by providing a means of converting data
- by supporting the definition of protocols for information exchange
- by reducing complexity in information processing applications
- by supporting content focus and dynamic presentation

This XML standards framework should be considered in light of functional areas of information management; including *data description* where the eXtensible Mark-up Language (**XML**), Document Type Definitions (**DTDs**), and **XML Schema** specifications play the key roles, *data transformation* where the eXtensible Stylesheet Language (**XSL**), and XSL Transformations (**XSLT**) specifications dominate (although they are not the only transformation mechanisms available to XML implementors), the representation of *data relationships* through the specifications of **XLink** and **XPointer**, and the *processing* of XML using the Document Object Model (**DOM**) or Simple API for XML[1] (**SAX**) mechanisms. Physical *data exchange* is predominantly the job of the application, although the Simple Object Access Protocol (**SOAP**) facilitates distributed object processing through particular services, XML data representation and standard network protocols. Other industry initiatives built on XML may also standardise information exchange.

The benefits of XML as a language for marking up data descriptively, over other languages such as HTML which restrict mark-up to representation or limited document-centric capabilities, are well documented.

Defining XML Resources

An XML document has two parts of description (similar in concept to object orientation – although XML itself is not object oriented); the document class and instances thereof. The key syntaxes for specifying classes of XML resources or documents are the DTD language and XML Schema.

[1] SAX is not a W3C initiative, rather it owes its evolution to those on the XML developers email list.

A DTD defines the structure of a class of XML resources. Unlike the Standard Generalised Mark-up Language (SGML), DTDs are optional in XML – depending on the validation requirement of the system. A XML document's structure is implemented as a single parent hierarchy (or inverted tree). The DTD specifies the logical structure (but not the semantics) of a document, the positions of elements and attributes, and the relationships between elements.

In order to be XML, a document must conform to the specification of the XML syntax in every way – in which case it is said to be *well-formed*. The checking of well-formedness is the first essential step in the XML *parsing* process. DTDs can be used by XML Parser software to also determine whether the data contained in an XML instance is structured correctly according to its class. This is the second step in the parsing process – essential to any XML processing application that requires confirmation of validity. Indeed, if a well-formed document instance conforms to its DTD, it is said to be a *valid* instance.

```
<?xml version="1.0"?>
<!DOCTYPE BookStore [
<!ELEMENT BookStore (Book)+>
<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
]>
<BookStore>
 <Book>
  <Title>3001 The Final Odyssey</Title>
  <Author>Arthur C Clarke</Author>
  <Date>1997</Date>
  <ISBN>0-246-12689-2</ISBN>
  <Publisher>Harper Collins</Publisher>
 </Book>
</BookStore>
```

XML Instance with Internal DTD Example

The above simple example illustrates a BookStore XML document together with its DTD. Amongst other problems, DTDs are simple to construct and in many cases are a perfectly adequate means of describing a document class, however they have a limited ability to describe the content of a document, very weak data typing (mostly textual) with little correspondence to application/database data types, an odd syntax, and limited structural expression.

XML Schema

The purpose of a XML Schema definition is to provide, more capably than DTDs, the structure and content model for a class of XML resources. XML Schemas are defined using XML syntax rather than a different 'legacy'[2] syntax. In fact XML Schema is an application of XML - a particular flavour of XML which gives us a set of defined elements and attributes that together supply mark-up for a specific purpose. XML Schema are understood by a type of processing technologies known collectively as *Schema Processors*. Similarly to the way XML is validated against a DTD in the traditional parsing process, XML instances can be validated against XML Schema definitions using XML Parsers that are integrated with Schema Processors.

Conceptually, a XML Schema is:

- A data model – you model how your data is to be represented in a XML instance document
- A contract - organisations can agree to structure their XML documents in conformance with a standard XML Schema
- A rich source of metadata, in addition to the instance data - data types, integrity constraints (ranges, limitations, formats, etc), relationships, etc

[2] By virtue of XML's evolution from the Standard Generalised Mark-up Language (SGML) – used primarily in the document management arena.

XML Schemas can be used to constrain not only document *structure* (through elements, attributes and namespaces) but also document *content* (datatypes, entities and notations). XML Schemas provide a more rigorous definition mechanism than DTDs for classes of XML resources, supporting definition of document structure, attributes, data-typing supporting inheritance, extensibility, substitutability, particular constraints, and so on. Crucially, XML Schemas facilitate XML data integrity checking using standard services.

XML Schema Definition

There are 3 parts to the W3C XML Schema specification:

- XML Schema Part 0: Primer - intended to provide an easily readable description of the XML Schema facilities.
- XML Schema Part 1: Structures – which defines the nature of XML Schemas and their component parts, provides an inventory of XML mark-up constructs with which to represent schemas, and defines the application of schemas to XML documents.
- XML Schema Part 2: Datatypes – which specifies datatypes that can be used in an XML Schema. These datatypes can be specified for element content that would be specified as #PCDATA and attribute values of various types in a DTD.

A Schema defines a content model within a global <schema> element, and consists of the necessary definitions for the permitted elements, their attributes, and any content constraints.

Note the following XML Schema definition (also known as a XSD) example and contrast it with the BookStore XML/DTD example described earlier. The XML syntax, in combination with the precise definitions and datatypes, represents a more capable foundation for structural and content verification using standard services.

```xml
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Title" type="xsd:string"/>
  <xsd:element name="Author" type="xsd:string"/>
  <xsd:element name="Date" type="xsd:string"/>
  <xsd:element name="ISBN" type="xsd:string"/>
  <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

XML Schema Example

When specifying element structure in a XSD, depending on the nature of the elements in question, some form of typing is required:

- Simple non-nested elements have a *simple* type (e.g. Title in the example above)
- Elements with attributes must have a *complex* type (examples later)
- Elements that embed other elements must have a complex type (e.g. Book in the example above)
- Attributes may only have simple types.

Annotation and additional structural control is achieved through special purpose Schema elements and attributes as defined in the specification.

Defining Data Types

XML Schema provides a set of primitive data types for elements and attributes, which are mapped to ‘standard’ application and database types. This set is influenced by the ISO 11404 standard on language-independent data types as well as the data types for SQL and Java. XML Schema also provides for custom type definitions that may be both 'simple' and 'complex'.

New simple types are derived from existing primitive or other simple types, and support constraining via ‘facets’. Complex types consist of simple or complex content with or without attributes. Facets are additional definitions related to custom simple types that restrict permitted values (such as length, range, and limitation through enumeration or regular expression patterns) in the corresponding XML instances. The following tables list the available primitive data types and corresponding facets.

Data Type	**maxInclusive**	**maxExclusive**	**minInclusive**	**minExclusive**	**totalDigits**	**fractionDigits**
byte	y	y	y	y	y	y
unsignedByte	y	y	y	y	y	y
integer	y	y	y	y	y	y
positiveInteger	y	y	y	y	y	y
negativeInteger	y	y	y	y	y	y
nonNegativeInteger	y	Y	y	y	y	y
nonPositiveInteger	y	Y	y	y	y	y
int	y	Y	y	y	y	y
unsignedInt	y	Y	y	y	y	y
long	y	Y	y	y	y	y
unsignedLong	y	Y	y			

XML Schema Primitive Data Types and Facets - Part 1

Data Type	Sample	length	minLength	maxLength	pattern	enumeration
string	Confirm this is electric	y	y	y	y	y
normalizedString	Confirm this is electric	y	y	y	y	y
token	Confirm this is electric	y	y	y	y	y
byte	-1, 126				y	y
unsignedByte	0, 126				y	y
base64Binary	GpM7	y	y	y	y	y
hexBinary	0FB7	y	y	y	y	y
integer	-126789, -1, 0, 1, 126789				y	y
positiveInteger	1, 126789				y	y
negativeInteger	-126789, -1				y	y
nonNegativeInteger	0, 1, 126789				y	y
nonPositiveInteger	-126789, -1, 0				y	y
int	-1, 126789675				y	y
unsignedInt	0, 1267896754				y	y
long	-1, 12678967543233				y	y
unsignedLong	0, 12678967543233				y	y
short	-1, 12678				y	y
unsignedShort	0, 12678				y	y
decimal	-1.23, 0, 123.4, 1000.00				y	y
float	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN				y	y
double	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN				y	y
boolean	true, false or 1, 0			y		y
time	13:20:00.000, 13:20:00.000-05:00				y	y
dateTime	1999-05-31T13:20:00.000-05:00				y	y
duration	P1Y2M3DT10H30M12.3S				y	y
date	31/05/1999				y	y
gMonth	--05--				y	y
gYear	1999				y	y
gYearMonth	1999-02				y	y
gDay	-31				y	y
gMonthDay	-26				y	y
Name	shipTo	y	y	y	y	y
QName	po:USAddress	y	y	y	y	y
NCName	USAddress	y	y	y	y	y
anyURI	http://www.example.com/, http://www.example.com/doc.html#ID5	y	y	y	y	y
language	en-GB, en-US, fr	y	y	y	y	y
ID		y	y	y	y	y
IDREF		y	y	y	y	y
IDREFS		y	y	y		y
ENTITY		y	y	y	y	y
ENTITIES		y	y	y		y
NOTATION		y	y	y	y	y
NMTOKEN	US, Brésil	y	y	y	y	y
NMTOKENS	US UK, Brésil Canada Mexique	y	y	y		y

XML Schema Primitive Data Types and Facets – Part 2

```
<xsd:simpleType name="TelephoneNumber">          ..1
   <xsd:restriction base="xsd:string">          ..2
      <xsd:length value="8"/>                  ..3
      <xsd:pattern value="\d{3}-\d{4}"/>        ..4
   </xsd:restriction>
</xsd:simpleType>
```

Creating a New Data Type

The above example illustrates the creation of a new simple data type TelephoneNumber which is based on a string primitive restricted through facets.

1. This creates a new datatype called 'TelephoneNumber'.
2. Elements of this type can hold string values,
3. But the string length must be exactly 8 characters long and
4. The string must follow the pattern: ddd-dddd, where 'd' represents a 'digit'.

Can you spot the redundancy in the definition[3]? The following example expands on our BookStore XML Schema example before, introducing additional precision for ISBNType.

[3] The length facet is redundant.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:simpleType name="ISBNType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{1}-\d{5}-\d{3}-\d{1}"/>
    <xsd:pattern value="\d{1}-\d{3}-\d{5}-\d{1}"/>
    <xsd:pattern value="\d{1}-\d{2}-\d{6}-\d{1}"/>
  </xsd:restriction>
 </xsd:simpleType>
 <xsd:element name="BookStore">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Book" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="Title" type="xsd:string"/>
            <xsd:element name="Author" type="xsd:string"/>
            <xsd:element name="Date" type="xsd:gYear"/>
            <xsd:element name="ISBN" type="ISBNType"/>
            <xsd:element name="Publisher" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Expanded XML Schema Example

XML Schema supports the concept of anonymous types, whereby the type definition is held locally to the element for which it is to be associated. This is useful for once-only requirements, whereas repeat-use of types should be catered for by using fully named type definitions. An example is shown below.

```
<xsd:element name="A" type="foo"/>
<xsd:complexType name="foo">
  <xsd:sequence>
    <xsd:element name="B" .../>
    <xsd:element name="C" .../>
  </xsd:sequence>
</xsd:complexType>

is equivalent to:

<xsd:element name="A">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="B" .../>
      <xsd:element name="C" .../>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Named vs. Anonymous Type Examples (Fragme nts)

Groups, Attributes, Lists and More

XML Schema also provides constraints that apply to groups of elements appearing in a content model, some of which you have already seen (such as sequence in the preceding examples):

- <group>
- <sequence>
- <choice>
- <all>

The above constraints do not apply to attributes, however attributes can be grouped for simplicity and reuse by using <attributeGroup>. The following example illustrates the definition of attributes for an element, in a similar way to the following DTD code:

```
<!ATTLIST Book
      Category (non-fiction | fiction) #REQUIRED
      InStock (true | false) "false"
      Reviewer CDATA " ">
```

```
<xsd:element name="BookStore">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Book"  maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="Title" type="xsd:string"/>
                        :…etc
          </xsd:sequence>
          <xsd:attribute Group ref="BookAttributes"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:attributeGroup name="BookAttributes">
  <xsd:attribute name="Category" use="required">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="non-fiction"/>
        <xsd:enumeration value="fiction"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="InStock" type="xsd:boolean" default="false"/>
  <xsd:attribute name="Reviewer" type="xsd:string" default=" "/>
</xsd:attributeGroup>
```

Attributes Example (Fragment)

XML Schema has three built-in list types: NMTOKENS , IDREFS , and ENTITIES. You can create new list types by derivation (possibly faceted) from existing atomic types. You cannot create list types from existing list types, nor from complex types. Note the following simple example and instantiation:

```
<xsd:simpleType name="ListOfNumbers">
  <xsd:list itemType="xsd:positiveInteger"/>
</xsd:simpleType>

<xsd:element name="LottoResults" type="ListOfNumbers"/>

  :
  : instantiated as…
  :

<LottoResults>3 6 21 24 34 40</LottoResults>
```

List Example (Fragments)

XML Schema also specifies many additional advanced features which are not discussed here, including Schema inclusion and redefinition, complex type extension and restriction, substitution groups, abstract elements and types, type derivation control, uniqueness (in conjunction with Xpath), Schema annotation, and more.

Referencing a Schema

The validation of a XML instance against a XML Schema may work in a number of ways. The XML Schema definition may be built-in to the application and referenced programmatically, or the Schema definition may be referenced as an external resource from the instance using the reserved xsi:schemaLocation attribute. This is similar in concept to the <!DOCTYPE … SYSTEM …> directive used with DTD-based validation. Either way, the Schema-enabled Parser should do the work. Centralised XML Schema repositories shared departmentally, at the enterprise level, or even at the industry level, are likely future additions to XML distributed architectures.

Some Schema Tools

The Schema specification was only ratified as a W3C recommendation in May 2001. As such, tools that purport to conform to the standard are few and far between. To quote: *"Although no hard figures have been circulated there is a general opinion that compatibility between implementations is far from satisfactory. A separate issue has been the conformance to the specifications… It would be unfortunate if XML documents end up being labelled as 'Best validated with…'"* [4].

The following tools and software are among the better examples currently available:

- **Validators and APIs**
  - Oracle XDK Schema Processor: http://technet.oracle.com/tech/xml/
  - Apache xerces: http://www.apache.org/xerces-j/
  - Microsoft MSXML4.0: http://www.microsoft.com
  - Sun Multi Schema Validator: http://wwws.sun.com/software/xml/developers/multischema/
  - XSV by Henry Thompson: ftp://ftp.cogsci.ed.ac.uk/pub/XSV/
  - IBM Schema Quality Checker: http://www.alphaworks.ibm.com/tech/xmlsqc

- **IDEs / GUI Oriented**
  - XML Spy: http://www.xmlspy.com
  - XML Authority: http://www.extensibility.com
  - SoftQuad XMetaL 3 : http://www.softquad.com
  - Oracle JDeveloper 9i: http://technet.oracle.com/products/jdev/

[4] Leigh Dodds "W3C XML Schema Needs You", xml.com, 27 March 2002

XML Schema and the Oracle XML Developer Kit (XDK)

The Oracle's XML Developer's Kit (XDK) 9i for Java, C, C++ and PL/SQL provides the following component technologies[5]:

- XML Parser/XSLT Processor
- XML Schema Processor
- XML Java Beans
- XML Class Generator
- XSQL Servlet
- XML SQL Utilities (XSU) for Java (and native XSU)
- Oracle Soap

As of v9.0.2.0.0A+, the XDK Schema Processor finally supported the W3C XML Schema recommendation. Prior to this version, the XDK Schema Processor was functional according to W3C working drafts and candidate recommendations. It is recommended that for W3C compliance reasons, and to mitigate interoperability issues with other XML services, that the 9i XDK platform be adopted as a starting point where operational Schema processing is required.

The Oracle XML Parser is a fully functional and standard-conformant validating Parser, which means that it can properly validate XML instances against internal and external DTDs, as well as providing implementations of DOM and SAX APIs, and integration with other XDK components (such as the XSLT Processor). Programmatically, parsing can be achieved as illustrated in the following simple Java example, which makes use of the XDK's DOM API :

```
try {
    DOMParser parser = new DOMParser();
    parser.setErrorStream(System.err);
    parser.setValidationMode(true);
    parser.showWarnings(true);
    parser.parse(createURL(fileName));
    XMLDocument doc = parser.getDocument();
    System.out.print("The elements are: ");
    NodeList nl = doc.getElementsByTagName("*");
    Node n;
     for (int i=0; i<nl.getLength(); i++) {
       n = nl.item(i);
       System.out.print(n.getNodeName() + " ");
     }
  }
  catch (Exception e) {
    System.out.println(e.toString());
  }
```

DTD validation in Java code using Oracle XML Parser (Fragment)

The XDK Schema Processor introduces several new objects into the mix:

- XSDBuilder - which verifies and builds XMLSchema objects (from supplied input XSDs)
- XMLSchema - Used by the XML Parser when it validates XML via the Schema Processor

The following code illustrates the Schema validation extensions in operation. Compare the code to the previous example:

[5] All components primarily for Java, with variable support for the other languages – check `http://technet.oracle.com` for current details.

```
try {
      XSDBuilder builder = new XSDBuilder;
      XMLSchema sch = (XMLSchema)builder.build(urlA);
      DOMParser dp = new DOMParser();
      dp.setErrorStream(System.err);
      dp.showWarnings(true);
      dp.setXMLSchema(sch);
      dp.setValidationMode(XMLParser.SCHEMA_VALIDATION);
      dp.parse(urlB);
      XMLDocument doc = dp.getDocument();
      System.out.print("The elements are: ");
      NodeList nl = doc.getElementsByTagName("*");
      Node n;
      for (int i=0; i<nl.getLength(); i++) {
        n = nl.item(i);
        System.out.print(n.getNodeName() + " ");
      }
   }
   catch (Exception e) {
      System.out.println(e.toString());
   }
```

Schema validation in Java code using Oracle XML Parser and Schema Processor (Fragment)

Other interesting possibilities exist for working with XML Schemas. It is possible to reverse engineer a XML Schema from (parts of) a database. For example one can use XML Spy to generate schemas from an ODBC-connected database. Oracle's XML SQL Utility (XSU) classes in the XDK can generate a DTD definition based on the structure of a query's result set, and it expected that this will also work for XML Schema in the near future. Oracle's Java Class Generator in the XDK can generate Java code stubs based on a Schema (or DTD) definition. Examples of such activity are well documented in Oracle's new documentation for XML development.

Final Comments and Web References

XML Schema, given its W3C status and rapid vendor adoption, should emerge as the pre-eminent XML modelling language and means of XML structure and content validation. XML Schema is complex however (and the subject of some debate), and there are some interesting alternatives undergoing independent development[6].

Oracle has been developing XML Schema capabilities throughout the specification's evolution to full recommendation. Their offerings in the XDK are amongst the best currently available and should be part of the developer's pallet of strong XML technology components, particularly if the Oracle 9i RDBMS is part of the overall application architecture.

Excellent Internet resources which can provide detailed technical, commercial and strategic information on XML Schema include:

- W3C XML Schema Specification - http://www.w3.org/TR/xmlschema-0/, http://www.w3.org/TR/xmlschema-1/, and http://www.w3.org/TR/xmlschema-2/
- Oracle Technology Network (OTN) - http://technet.oracle.com/tech/xml/
- The XML Cover Pages - http://xml.coverpages.org/
- The XML Schema developers email list service - xmlschema-dev-request@w3.org

[6] Including "Relax"- which is positioned as being radically simpler and upwardly compatible to XML Schema, "TREX" – developed by James Clark, lead author of the XSLT and XPath specs, and "Schematron" – which is XPath based, requires only XSLT to run, and aims to be complementary to XML Schema.

About the Author

Dennis Remmer holds degrees in computer science and geospatial information systems (GIS) from the University of Queensland, Brisbane. His time with Unisys Corporation, ARC Systems, the Distributed Systems Technology Centre (DSTC), the Collaborative Health Informatics Centre (CHIC), and in recent years building a specialist company (E2), has led to substantial commercial and international experience in systems development and professional development. His primary areas of technology expertise include Java, XML, Web, SQL, database, multimedia and GIS technologies.

He co-authored a book ("Java Thin-Client Programming for a Network Computing Environment") for IBM and Prentice Hall, instructed numerous advanced courses, and has spoken at many conferences and seminars; most recently on XML on behalf of Oracle throughout Scandinavia and the Asia/Pacific region, and on database systems on behalf of the United Nations at a summer school for developing nations held in Ulaanbaatar, Mongolia. He is the current President of the Australian Oracle User Group, and runs a record label in his spare time, specialising in electronic and contemporary music production.

Acknowledgement

David Jackson, Bob Brown, and Robert Costello are particularly acknowledged for their contribution, collaboration and/or influential previous efforts in this space.

Dennis Remmer
E2, Australia
dennis@etwo.com.au