Electronic Access Conference
San Diego, California
New Orleans, Louisiana
2003

Session 14

XML: Advanced Guide

Holly A. Hyland, FSA
Andrew Smalera, XML Framework

Agenda

- Objectives
- What is the XML Framework at FSA?
- XML Usage at FSA
- XML Usage in the Community
- XML Schema Design Best Practices
- XML Supporting Technologies
  - XML Development Tools
  - XML Parsers
  - XSLT, XQuery, XLink, and XPath
- References
- Questions

Electronic Access Conference
San Diego, California
New Orleans, Louisiana
2003

Objectives

- Provide an overview of the FSA XML Framework.
- Provide an update on XML Usage at FSA and within the Community
- Present XML Schema Design Patterns.
- Present information on XML supporting technologies including:
  - Development Tools
  - XML Parsers
  - XSLT, XQuery, XPath, and XLink
- Provide a list of references for additional research.

What is the FSA XML Framework?

This session provides an overview of the current status of XML usage within the Financial Aid Community and some advanced topics relevant to implementing and taking advantage of some XML's benefits.

Electronic Access Conference
San Diego, California
New Orleans, Louisiana
2003

XML Usage at FSA

- Expanded use of XML by FSA
  - More schools and software vendors are moving to being COD Full Participants.
  - EdExpress is incorporating XML support.
  - The CPS ISIR will be implemented as an XML Schema for the 05-06 Award Year.

Electronic Access Conference
San Diego, California
New Orleans, Louisiana
2003

XML Usage in the Community

- Expanded use of XML by the Financial Aid Community
  - The Common Record: CommonLine XML Schema has been drafted and is in the process of being implemented.
  - The Postsecondary Academic Transcript has been drafted as an XML Schema.
  - METEOR and ELM are using and providing support for XML.

Electronic Access Conference
San Diego, California
New Orleans, Louisiana
2003

XML Schema Design Best Practices

- Overview
- Russian Doll Design
- Salami Slice Design
- Venetian Blind Design

Electronic Access Conference
San Diego, California
New Orleans, Louisiana
2003

XML Schema Design Best Practices: Overview

- While FSA and the Financial Aid Community have developed a number of XML Schemas that are in use today, individual schools and vendors may find a use for developing their own schemas for internal data exchange and processing.
- XML Schema Design Best Practices provides information on the three design patterns commonly used to create XML Schemas.
- Each design pattern has its own pros and cons and may be used depending on the situation.
- An understanding of these different design patterns will be helpful for schools and vendors to provide feedback on future Schema development efforts by FSA and the Community.

Electronic Access Conference
San Diego, California
New Orleans, Louisiana
2003

XML Schema Design Best Practices: Russian Doll Design

- The Russian Doll Design defines objects in local scope.
- Elements created using this methodology will have Schemas that are very similar to the instance documents.
  - Limits the reusability of Schema designs.
- Facilitates hiding namespaces.
  - Can prevent namespace issues like name collisions.

Example Russian Doll Design Schema Snippet

```
<xsd:element name="Movie">
   <xsd:complexType>
        <xsd:sequence>
               <xsd:element name="Title" type="xsd:string"/>
               <xsd:element name="Director" type="xsd:string"/>
               <xsd:element name="Genre" type="xsd:string"/>
               <xsd:element name="ReleaseYear" type="xsd:gYear"/>
         </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

XML Schema Design Best Practices: Salami Slice Design

- The Salami Slice Design defines all objects in the global scope.
- Elements created using this methodology make object reuse very easy.
- Mapping between the Schema and an instance document will not be as straight forward.
  - It should be noted that this limitation does not carry over to automated validation of instance documents against Schemas.
- Allows the reuse of elements so that Schema designers must be cognizant of possible namespace issues like name collisions.

Example Salami Slice Design Schema Snippet

```
<xsd:element name="Movie">
  <xsd:complexType>
      <xsd:sequence>
            <xsd:element ref="Title"/>
            <xsd:element ref="Director"/>
            <xsd:element ref="Genre"/>
            <xsd:element ref="ReleaseYear"/>
      </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="Title" type="xsd:string"/>
<xsd:element name="Director" type="xsd:string"/>
<xsd:element name="Genre" type="xsd:string"/>
<xsd:element name="ReleaseYear" type="xsd:gYear"/>
```

XML Schema Design Best Practices: Venetian Blind Design

- The Venetian Blind Design leverages the design advantages of both the Russian Doll and Salami Slice Designs.
- Facilitates reuse while also hiding namespace complexities (by creating type definitions).
- Instead of actually creating elements and referencing them, a Schema designer would create a type, and reference that when creating their elements.

Example Venetian Blind Design Schema Snippet

```
<xsd:simpleType name="TitleType">
   <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:simpleType name="DirectorType">
   <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:simpleType name="GenreType">
   <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:simpleType name="ReleaseYearType">
   <xsd:restriction base="xsd:gYear"/>
</xsd:simpleType>
<xsd:complexType name="MovieType">
<xsd:sequence>
   <xsd:element name="Title" type="TitleType"/>
   <xsd:element name="Director" type="DirectorType"/>
   <xsd:element name="Genre" type="GenreType"/>
   <xsd:element name="ReleaseYear" type="ReleaseYearType"/>
</xsd:sequence>
</xsd:complexType>
<xsd:element name="Movie" type="MovieType"/>
```

XML Supporting Technologies

- Development Tools
- XML Java Parsers
- XSLT
- XQuery
- XPath
- XLink

Electronic Access Conference
San Diego, California
New Orleans, Louisiana
2003

XML Supporting Technologies: Development Tools

- There are many categories of tools and many different tools available to support XML development. The following are representative Integrated Development Environment (IDE) tools:

- Sonic **Stylus Studio** provides support for authoring XQuery, XSLT stylesheets, XML schema, and related XML documents.
- Altova **XMLSpy** is an XML Development Environment that can be used for designing, editing, and implementing XML. It provides a graphical view of schemas and instance documents. In addition, XMLSpy provides an integrated XML instance document validator.
- Tibco **TurboXML** is an IDE for developing and managing XML assets. It provides support for creating, validating, converting, and managing XML schemas, XML files, and DTDs.

Electronic Access Conference
San Diego, California
New Orleans, Louisiana
2003

XML Supporting Technologies: XML Java Parsers

- Parsers allow you to read in XML documents. Provide access to information stored in XML documents.
- There are three general categories of parsers:
  - **<u>All-in-Memory Parsers</u>** – load the entire XML document into memory and provide a tree-like view of the document. As a result, the entire document must be processed before you can access any piece of the document at all.
  - **<u>Push Parsers</u>** – hide the interaction with the actual document and "push" the tokens to user through callback methods. Reads the XML stream, and when it encounters an element (or entity, etc.), it generates an event. It is up to the application to handle those events, usually via a callback or an event handler class.
    - SAX
    - DOM
  - **<u>Pull Parsers</u>** – the application developer is responsible for the parsing loop, pulling elements (or entities, etc.) out of the XML stream explicitly.
- Currently, the two most widespread parsers are DOM and SAX.

XML Java Parsers: Document Object Model (DOM)

- The Document Object Model (DOM) API is associated with all-in-memory parsers.
- DOM is more than just a parsing technology; it is a generic document object model.
- DOM can read in an XML stream, can optionally validate it against a schema or DTD, and when it's done parsing, it provides a tree view of the document.
- DOM provides access to the information stored in an XML document as a hierarchical object model. DOM creates a tree of nodes (based on the structure and information in the XML document) and provide access to the information by interacting with the tree of nodes.
- In DOM, each element corresponds to a node in the tree. Using DOM, developers can manipulate the document in any number of ways, including addition, modification, and removal of nodes and text content.
- A DOM instance can also be serialized to an XML stream.
- There is a great deal of overhead to using DOM, and for very large documents that can be a big problem. For each element, there must be a corresponding node instance in memory, as well as a collection to hold any attributes that the node may contain. DOM stores everything, including attribute values, as a string. Thus, if you have Boolean or numerical attributes, DOM wastes space storing them as strings.
- DOM will usually be outperformed by a streaming parser, because most DOM implementations are now built on top of a streaming parser.
- The last major performance issues it the overhead of creating and maintaining the tree structure that makes DOM so useful in the first place.

XML Java Parsers: Simple API for XML (SAX)

- Simple API for XML (SAX) Parser
- The SAX API is associated with push parsers.
- SAX allows users to define a set of handler objects through which it notifies you when interesting events occur during the sequential parsing of a document.
- Since it is forward-only, SAX is not particularly resource intensive.
- Since SAX basically just reads XML content from a stream and then passes that content on to the application through the event handler interfaces, SAX implementations generally have very little overhead, which in turn usually leads to good parsing performance.
- If it is not implemented properly, the event-handling code for dealing with complex or deeply nested documents can become very convoluted and difficult to read and maintain.
- SAX doesn't define an object model of its own. Therefore, in most cases the developer will have to define their own data structures to store the data.
- Once the SAX implementation has commenced the parsing process, the only way to terminate the process is to throw an exception, which is less than ideal.

XML Supporting Technologies: XSLT

- XSLT provides developers with a higher-level language to access and transform XML streams. There are two methods for implementing XSLT:
  - **Push** – The stylesheet provides rules for mapping input streams to target streams. Rules are set up for each element that will be encountered and "Pushed" through the transformation.
  - **Pull** – The stylesheet provides rules for how elements are pulled out of a source stream, put into the target stream.

XML Supporting Technologies: XQuery

- XQuery enables users to query and extract data from an XML document. It is an extension of XPath, but has manipulation capabilities rather than being just a lookup.
- XML is increasingly being used to model and store structured, semi-structured and relational data, and XQuery provides a powerful mechanism to access and manipulate the data stored in XML documents.
- For additional information refer to: http://www.w3.org/TR/xquery/

XML Supporting Technologies: XPath

- XPath is a non-XML language used to identify parts of XML documents. It does this by viewing the hierarchical structure of an XML document as a tree of nodes and returns results based on the position of a node, its type or content.
- XSLT and XPointer use XPath.
- For additional information refer to: http://www.w3.org/TR/xquery/

Electronic Access Conference
San Diego, California
New Orleans, Louisiana
2003

XML Supporting Technologies: XLink

- XLink allows elements to be inserted into XML documents in order to create and describe links between resources.
- There are two types of links – simple and extended.
  - Simple links are identical to HTML links enabling the linking to other HTML and XML documents.
  - Extending links allow two-way linking between a group of documents and menu capabilities.
- For additional information refer to: http://www.w3.org/TR/xlink/

Electronic Access Conference
San Diego, California
New Orleans, Louisiana
2003

XML References

Additional information on XML can be found in the following references:

- www.w3c.org
- www.ebxml.org
- www.oasis-open.org
- www.xml.com
- XML Journal <http://sys-con.com/xml/>
- www.xfront.com

Questions?

We appreciate your feedback and comments. We can be reached at:

Name: Holly A. Hyland
Phone: 202-377-3710
Email: Holly.Hyland@ed.gov

Name: Andrew Smalera
Phone: 202-962-0789
Email: Andrew.Smalera@accenture.com