Query XML Data with .NET

<BrianNoyes/>

www.idesign.net
brian.noyes@idesign.net

About Brian

- Microsoft MVP in ASP.NET
- Writing
  - MSDN Magazine, asp.netPRO, Visual Studio Magazine, .NET Developer’s Journal
  - Building Windows Forms Data Applications with .NET 2.0, Addison-Wesley, expected release spring 2005
- Speaking
  - Microsoft TechEd, Visual Studio Connections, DevEssentials, VSLive!, INETA Speakers Bureau
- Participates in Microsoft design reviews
- E-mail: brian.noyes@idesign.net
- Blog: http://www.softinsight.com/bnoyes
About IDesign

- .NET Architecture and Design Consulting and Training
  - On site architecture and design consulting and mentoring
  - .NET Master Class and other training courses
- Top industry .NET architects, speakers, writers
- Site Resources
  - Sample architecture document
  - C# Coding Standard
  - Article links
  - Code samples
- Visit us at www.idesign.net

It’s A User Group Thing

- As an INETA member, your User Group gets:
  - Visits from Speaker Bureau members at no charge
  - INETA even pays for the pizza
  - Discounts on software, conferences, subscriptions and more
  - Newsletter, content repository and forums of use to User Group members
- INETA needs volunteers
  - Benefit your User Group and yourself by getting involved
- INETA Web Site: www.ineta.org
Gameplan

- Quick Intro to XPath
- Loading XML
- Query DOM Documents
- Working with XPathDocuments

XML Syntax Rules

- XML Syntax Rules
  - Case sensitive
  - A document is composed of nodes
    - Everything in a document is a node
  - A document has one and only one root node
    - That root node is an element
  - Elements have both start and end tags
  - Elements are properly nested
  - Elements can contain attributes, child elements, or text
  - Special node definitions for specific purposes:
    - Comments, CDATA, Processing Instructions, Entities
Sample XML Document

```xml
<?xml version="1.0" encoding="utf-8" ?>
<!-- The Music element is the root node -->
<Music>
    <Artist name="Evanescense">
        <Album name="Fallen">
            <Track number="1">Going Under</Track>
            <Track number="2">Bring Me To Life</Track>
            <Track number="3">Everybody's Fool</Track>
        </Album>
    </Artist>
    <Artist name="Disturbed">
        <Album name="Believe">
        </Album>
    </Artist>
</Music>
```

XML Namespaces

- Used to scope named objects in a document
- Similar to programming language namespaces
- Consist of an arbitrary string
- Should be unique to ensure no name clashes with other XML documents
  - Two conventions exist:
    - URL: Used because domain names are guaranteed to be unique in the Internet space
      `xmlns="http://www.idesign.net/DataAccess"`
    - URN: arbitrary naming with specific syntax
      `xmlns="urn:idesign-net"`
XML Namespaces

- Can be declared at document or element scope
  - Child nodes scoped to the default namespace of their parent node
- Prefixes used to allow scoping to multiple namespaces with shorthand notation

```xml
<?xml version="1.0" encoding="utf-8" ?>
<Root xmlns="http://www.idesign.net/DataAccess"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:foo="urn:yadda-foobar-skwee">
  <!-- Unscoped elements are assumed to be defined in http://www.idesign.net/DataAccess namespace -->
  <GrandParent foo:myattribute="hello">
    <Parent xmlns="urn:idesign-net" xmlns:yadda="urn:yadda">
      <!-- Now unscoped elements are assumed to be defined in urn:idesign-net namespace -->
      <Child>
        <yadda:SomeElement>DeDoDoDoDeDaDaDa</yadda:SomeElement>
      </Child>
    </Parent>
  </GrandParent>
</Root>
```

Commonly Used XML Grammars in .NET

- XML Schemas
  - Used to define the structure / type system of an XML document
- XPath
  - Used for querying XML documents
- XSLT
  - Used to transform XML into other document formats
XML Schemas

XML Schemas define:
- Allowable structure of an XML document
  - What elements and attributes
  - Relative location of elements and attributes
- The type system of the XML document
  - Organization of data in the document
  - The types of data contained by nodes
  - XML Documents are instances of a particular schema
- The “shape” of the document

Three generations to contend with:
- Document Type Definitions (DTDs)
- XML Data Reduced (XDR)
- XML Schema 1.0 Definition Language (XSD)
Document Type Definitions

- Structural definition of an XML document
- Not XML itself
- Limited type definition capability
- Quickly becoming obsolete

```xml
<!DOCTYPE Music [ 
<!ELEMENT Music (Artist*)>
<!ELEMENT Artist (Album*)>
<!ELEMENT Album (Track*)>
<!ATTLIST name CDATA #REQUIRED>
<!ELEMENT Track (#PCDATA)>
<!ATTLIST number CDATA "">
]>
```

XML Data Reduced (XDR)

- XML Schema Definition Language (XDR)
  - Early attempt by Microsoft to standardize an XML-based schema language
  - Based on the W3C Schema Recommendation before W3C approval
  - Not as expressive as XML Schema 1.0

```xml
<?xml version="1.0" encoding="utf-8" ?>
<Schema xmlns="urn:idesign.net-music" xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <AttributeType name="number" dt:type="int" />
  <AttributeType name="name" dt:type="string" />
  <ElementType name="Track" content="mixed"> 
    <attribute type="name" />
  </ElementType>
  <ElementType name="Album">
    <element type="Track" />
    <attribute type="name" />
  </ElementType>
  <ElementType name="Artist">
    <element type="Album" />
    <attribute type="name" />
    <Element name="Track" content="mixed">
      <attribute type="name" />
    </Element>
  </ElementType>
  <ElementType name="Music">
    <element type="Artist" minOccurs="0" maxOccurs="*" />
  </ElementType>
</Schema>
```
XML Schema 1.0 (XSD)

- The formal recommendation approved by W3C
- Includes:
  - Primitive type definitions
  - Simple and complex type definitions
  - Element and attribute declarations
  - Namespace scoping
  - Inheritance, includes, imports
- The default schema mechanism used by .NET

XML Schema 1.0 Example

```xml
  <xs:element name="Music" msdata:isDataSet="true" msdata:enforceConstraints="false">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="Album" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence name="Track" minOccurs="0" maxOccurs="unbounded">
              <xs:element name="Track" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence name="Track_Text" minOccurs="0" maxOccurs="unbounded">
                    <xs:element name="Track_Text" minOccurs="0" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:sequence name="Track_Text" minOccurs="0" maxOccurs="unbounded">
                          <xs:element name="Track_Text" minOccurs="0" maxOccurs="unbounded">
                            <xs:complexType>
                              <xs:sequence name="Track_Text" minOccurs="0" maxOccurs="unbounded">
                                <xs:element name="Track_Text" minOccurs="0" maxOccurs="unbounded">
                                  <xs:complexType>
                                    <xs:sequence name="track_text" minOccurs="0" maxOccurs="unbounded">
                                      <xs:element name="track_text" minOccurs="0" maxOccurs="unbounded">
                                        <xs:complexType>
                                          <xs:sequence name="track_text" minOccurs="0" maxOccurs="unbounded">
                                            <xs:element name="track_text" minOccurs="0" maxOccurs="unbounded">
                                              <xs:complexType>
                                                <xs:sequence name="track_text" minOccurs="0" maxOccurs="unbounded">
                                                  <xs:element name="track_text" minOccurs="0" maxOccurs="unbounded">
                                                    <xs:complexType>
                                                      <xs:sequence name="track_text" minOccurs="0" maxOccurs="unbounded">
                                                        <xs:element name="track_text" minOccurs="0" maxOccurs="unbounded">
                                                          <xs:complexType>
                                                            <xs:sequence name="Track_Text" minOccurs="0" maxOccurs="unbounded">
                                                              <xs:element name="Track_Text" minOccurs="0" maxOccurs="unbounded">
                                                                <xs:complexType>
                                                                  <xs:sequence name="Track_Text" minOccurs="0" maxOccurs="unbounded">
                                                                    <xs:element name="Track_Text" minOccurs="0" maxOccurs="unbounded">
                                                                      <xs:complexType>
                                                                        <xs:sequence name="Track_Text" minOccurs="0" maxOccurs="unbounded">
                                                                          <xs:element name="Track_Text" minOccurs="0" maxOccurs="unbounded">
                                                                            <xs:complexType>
                                                                              <xs:sequence name="Track_Text" minOccurs="0" maxOccurs="unbounded">
                                                                                <xs:element name="Track_Text" minOccurs="0" maxOccurs="unbounded">
                                                                                  <xs:complexType>
                                                                                    <xs:sequence name="Track_Text" minOccurs="0" maxOccurs="unbounded">
                                                                                      <xs:element name="Track_Text" minOccurs="0" maxOccurs="unbounded">
                                                                                      </xs:complexType>
                                                                                </xs:complexType>
                                                                              </xs:complexType>
                                                                            </xs:complexType>
                                                                          </xs:complexType>
                                                                        </xs:complexType>
                                                                      </xs:complexType>
                                                                    </xs:complexType>
                                                                  </xs:complexType>
                                                                </xs:complexType>
                                                              </xs:complexType>
                                                            </xs:complexType>
                                                          </xs:complexType>
                                                        </xs:complexType>
                                                      </xs:complexType>
                                                    </xs:complexType>
                                                  </xs:complexType>
                                                </xs:complexType>
                                              </xs:complexType>
                                            </xs:complexType>
                                          </xs:complexType>
                                        </xs:complexType>
                                      </xs:complexType>
                                    </xs:complexType>
                                  </xs:complexType>
                                </xs:complexType>
                              </xs:complexType>
                            </xs:complexType>
                          </xs:complexType>
                        </xs:complexType>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
**XPath Overview**

- Query Syntax for XML
  - Somewhat like SQL for XML
    - XQuery is closer to SQL capabilities
  - Common syntax and semantics for location specifications in a document
  - Navigates the hierarchical structure of a document
  - Represents a document as a tree of nodes

**XPath Expressions**

- Evaluate to a node set, boolean, number, or string
- Expressions are evaluated based on a current context or position within a document
  - Kind of like a cursor in a database
- Not XML themselves
XPath Expression Composition

- XPath Expressions are composed of location steps
  - Each location step has
    - Axis (optional/implied)
    - Node Test
    - Predicate
  - Results from one location step used as the context node for the next location step
  - Location steps are separated by slashes

locstep1/locstep2/locstep3/...

XPath Expression Composition

- Axes to define a relative path
  - descendant::, child::, parent::, etc.
  - Implicit child:: if not provided
  - Shorthand operators for common axes:
    - / - root
    - // - descendant
    - . - current node
    - .. - parent node
XPath Expression Composition

- Node Test defines a node to match against
  - Element, attribute, built-in function
- Predicate defines conditional logic to determine matches
  - Can define multiple predicates
    - Implicit AND of separate predicates
  - Can define complex predicates
    - Boolean logic, built-in functions, extension functions

XPath and Namespaces

- Nodes frequently scoped to a namespace
- Up to the processor to decide what to do about it
  - Usually only matches against properly scoped node names with namespaces
- Can include namespace prefixes in node names
  - Processor has to be made aware of the namespace URI
    - Use XmlNamespaceManager in .NET
Sample XML Document

<?xml version="1.0" encoding="utf-8" ?>
<!-- The Music element is the root node -->
<Music>
  <Artist name="Evanescense">
    <Album name="Fallen">
      <Track number="1">Going Under</Track>
      <Track number="2">Bring Me To Life</Track>
      <Track number="3">Everybody's Fool</Track>
    </Album>
  </Artist>
  <Artist name="Disturbed">
    <Album name="Believe">
    </Album>
  </Artist>
</Music>

XPath Examples

Examples:
Music/Artist
Music/Artist/@name
/Music/Artist/Album[@name='Fallen']
descendant::Album[@name='Fallen']
descendant::Album/@name
/Music/child::Artist
descendant::*[parent::Album]
descendant::*[parent::Album][@number = 2]
descendant::*[parent::Album][@number = 2][contains(text(),'Life')]
descendant::*[parent::myns:Album][@number = 2]
Loading XML

- **Document Object Model (DOM) Oriented**
  - XmlDocument.Load()
    - Load from disk or stream
  - XmlDocument.LoadXml()
    - Load from a string representation of XML document

- **Data Oriented**
  - DataSet.ReadXml()
    - Loads XML that is suitable for representation as a DataSet
    - Will infer a schema if one is not present
  - XmlDataDocument constructor
    - Takes a DataSet and provides an XmlDocument derived class

Loading XML

- **XPath Oriented**
  - XPathDocument() constructor
    - Load from stream, uri, reader

- **XSLT Oriented**
  - XslTransform.Load()
    - Load from URL

- **Stream Oriented**
  - Stream + XmlTextReader
    - Read only/forward only node access from a stream
  - Document + XmlNodeReader
    - Read only/forward only node access from a document
Performing DOM Queries

- XmlNode class methods
  - SelectNodes()
  - SelectSingleNode()

- XmlDocument class methods
  - GetElementsByTagName()
  - GetElementById()

XmlNode.SelectNodes()

- Easiest queries when working with DOM documents
- Takes an XPath expression
- Optional: takes an XmlNamespaceManager to scope namespace prefixed node names
- Returns an XmlNodeList
  - Iterate the results and operate on them
    - Provides Read/Write access to nodes in the DOM
  - Abstract base class – actually an XPathNodeList instance
- Uses an XPathNavigator under the covers
XmlNode.SelectNodes()

```csharp
// Load the document
XmlNode doc = new XmlDocument();
doc.Load("C:\demos\MusicBase.xml");

// Call SelectNodes with a valid XPath Query
XmlNodeList nodes = doc.SelectNodes(
    "descendant::Track[@number=2]/text()";
foreach (XmlNode node in nodes)
{
    Console.WriteLine(node.Value);
}
```

XmlNode.SelectSingleNode()

- Used to obtain a single match for a query
- Actually just calls SelectNodes and returns first node in list
  - Not very efficient, query could return a large result set
  - Just a convenience method
- Improving performance: use smarter queries
  - Construct an XPath query that will only return a single node
    - Smart knowledge of the target data
    - [position() = 1] predicate
XmlNode.SelectSingleNode()

// Load the document
XmlDocument doc = new XmlDocument();
doc.Load("C:\demos\MusicBase.xml");
// Select a single node
XmlNode dumb = doc.SelectSingleNode("descendant::Track[@number=2]");
XmlNode smart = doc.SelectSingleNode("descendant::Track[@number=2][position()=1]);
Console.WriteLine(dumb.InnerText);
Console.WriteLine(smart.InnerText);

Other DOM Query Options

- XmlDocument.GetElementsByTagName()
  - Gets collections of elements
  - Uses internal element tables to obtain
  - Cannot control level of element selected in heirarchy
- XmlDocument.GetElementByID()
  - Takes an ID value
  - Returns the element whose ID attribute matches
  - Only works with DTD schema definition associated with document
  - Not very useful
**XmlDataDocument**

- Bridges the XML and relational views of data
  - Easy access to XML functionality on the data
    - Queries, transformation, validation
  - Easy access to relational functionality with the data
    - .NET Databinding
- Derives from XmlDocument
  - "Is a" XmlDocument
  - Looks like a DataSet when needed through a property
- Everything covered about XmlDocument applies
  - It's just a derived class
  - Has a specialized XPathNavigator implementation

**Enter the .NET XPath Processing Engine**

- New XML nav and query model
- Other models good for what they were designed for
  - DOM is heavyweight, read/write, random access, handles all XML node types
    - Rich access to complex object model
  - XmlReader is lightweight, read-only, forward-only, handles all XML Node Types
    - Fast serialized access for getting data in memory
- Need something that gives the best of both worlds
**XPathNavigator**

- Abstract base class used to define XML navigation and query API for XML in .NET
  - Document classes that support implement IXPathNavigable
- Lightweight object model
  - Hierarchical tree based
  - Restricted to just elements, attributes, namespaces
    - Name and value

**XPathNavigator**

- Read-Only
- Random-Access
  - Can move forward or backward in document, traverse up or down levels easily
**XPathDocument**

- Lightweight document class
- Load XML into memory for use with XPathNavigator
- Constructor takes stream, URL, or reader
- Much lighter weight object model than XmlDocument
- Get XPathNavigator with CreateNavigator()
  - Actually an XPathDocumentNavigator instance

**XPath Objects**
XPathNavigator Methods

- MoveXXX Methods
  - Forward/backward navigation through nodes
- SelectXXX Methods
  - Select()
    - primary XPath query evaluation method
    - Returns iterator for node set results
  - SelectAncenstors()/SelectChildren()/SelectDescendants()
    - Easier selection along particular axes

- Compile()
  - Pre-compile XPathExpression for rapid repeated execution
- Evaluate()
  - Similar to Select(), but works for queries that do not return node sets (string, bool, number)
- Clone()
  - Get a copy of the iterator for navigating child nodes
- GetAttribute()/GetNamespace()
  - access node info
- Positional checks
XPathNavigator Properties

- NodeType
- Name – fully qualified
- LocalName – no namespace
- NamespaceURI
- Value
- Prefix – namespace prefix
- NameTable
- HasAttributes
- HasChildren
- Etc.

XPathExpressions

- Class to support declaring and compiling XPath queries
- Not created directly, returned from XPathNavigator.Compile()
- Only way to associate namespace information with a query
**XPathNodeIterators**

- What gets returned from a call to Select()
- Lightweight iterator on the resulting node-set
- Access the current node with Current property
- Loop through with MoveNext()
- Count property is pre-calculated

---

**Querying with XPathNavigator**

- Load the XML into a document class
- Get an XPathNavigator for the document
- Optionally compile an XPath expression for reuse
- Call Select() to get an XPathNodeIterator
  or
- Call Evaluate() to get a value back
  - Returned as an object, must cast to the expected type
  - Can also return node set iterators this way too
Simple XPathNavigator Query

```csharp
// Load the document object
XPathDocument doc = new XPathDocument("C:\\demos\\MusicBase.xml");
// Get a navigator for the document
XPathNavigator nav = doc.CreateNavigator();
// Perform the query
XPathNodeIterator iter = nav.Select("descendant::Track[@number=2]");
// Move through the results with the iterator
while (iter.MoveNext())
{
    Console.WriteLine(iter.Current.Value);
}
```

Working with Namespaces

- Create an XmlNamespaceManager using the NameTable of the navigator
- Add namespace prefix/URI combos
- Compile the XPath expression
- Set the XPathExpression context
Working with Namespace

```csharp
// Load the document
XPathDocument doc = new XPathDocument("C:\demos\MusicDefNs.xml");
// Get a navigator for the document
XPathNavigator nav = doc.CreateNavigator();
// Compile the query with namespace prefixes
XPathExpression exp = nav.Compile("descendant::foo:Track[@number=2]");
// Create the namespace manager
XmlElementManager mgr = new XmlNamespaceManager(nav.NameTable);
// Add an alias for the default namespace
mgr.AddNamespace("foo","urn:foo");
// Set the context on the expression
exp.SetContext(mgr);
// Perform the query
XPathNodeIterator iter = nav.Select(exp);
// Loop through the results
while (iter.MoveNext())
{
    Console.WriteLine(iter.Current.Value);
}
```

Coming in .NET 2.0

- XPathEditor
  - Derived from XPathNavigator
  - Allows you to edit nodes
- XPathChangeNavigator
  - Derived from XPathNavigator
  - Allows you to navigate modified nodes

http://weblogs.asp.net/dareobasanjo/archive/2004/10/13/241591.aspx
Summary

- Prefer XPathDocument/XPathNavigator when you don't need to modify nodes
- Use XmlDocument for read/write nodes
- Use XmlDataDocument for hierarchical and relational views of data

Resources:
- Applied XML Programming for Microsoft .NET, Dino Esposito, Microsoft Press
- Manipulate XML Data Easily with the XPath and XSLT APIs in the .NET Framework, Dino Esposito, MSDN Magazine, July 2003
  - [http://www.msdn.microsoft.com/msdnmag/issues/03/07/XPathandXSLT/default.aspx](http://www.msdn.microsoft.com/msdnmag/issues/03/07/XPathandXSLT/default.aspx)

Questions/comments: brian.noyes@idesign.net