Generating XForms applications using the National Information Exchange Model (NIEM)

Creating semantically precise XForms directly from exchange documents

Dan McCreary August 21, 2007

The National Information Exchange Model (NIEM) is a United States' federal data standard for the exchange of XML data between federal, state, and local organizations. The NIEM provides a rich set of universal data elements that non-programmers can use to build semantically precise data exchanges among organizations. This article demonstrates how XForms applications can be automatically created from a National Information Exchange Model (NIEM) constraint schema, and shows how graphical tools can allow non-programmers to automatically create rich Web applications using a model-driven approach. It gives an example of how a short XML transformation (XSLT) is used to achieve this task and how the transformation can be modified and extended by developers.

Before you begin

This article discusses XML Schema, XSLT, and XForms standards. Familiarity with XML and other W3C standards is useful. To run the example an XSLT 2.0 XML transformer such as Saxon is needed. The examples also use the Firefox XForms add-on to render the XForms.

Introduction

Business Unit Empowerment is a growing strategy in organizations that strive to lower the cost of developing information systems. The strategy allows non-programmers in each business unit to create and maintain applications without the need for central IT software developers and has become an emphasis to lower IT development costs.

Central to this movement is the ability to create rich Web forms that generate and update complex data. Previously, the creation of rich Web forms required teams of programmers with extensive programming skills. Today, however, the rise of declarative systems is changing the programming landscape. Unlike procedural systems, declarative systems use a set of small languages and graphical tools to build complex applications without the need for programmers. The declarative approach is one of the principal methods organizations are implementing to lower their Web-related development costs.

This article focuses on four small systems: NIEM, XML Schema, XSLT, and XForms. These systems combined with a native XML database (such as IBM® DB2® Version 9 "pureXML"™) enables an organization to produce rich Web applications with few software developers.

The central strategy in this process is to equip subject-matter experts (SME) and business analysts (BAs) with tools to select from constrained lists of items, drawn from controlled vocabularies, and draw pictures that precisely capture business requirements. This supports the general trend toward data selection over procedural programming to lower overall IT development and maintenance costs. In this example we use the NIEM, although any controlled vocabulary with sub-schema generation tools can be used.

Overview of the NIEM and sub-schema generation

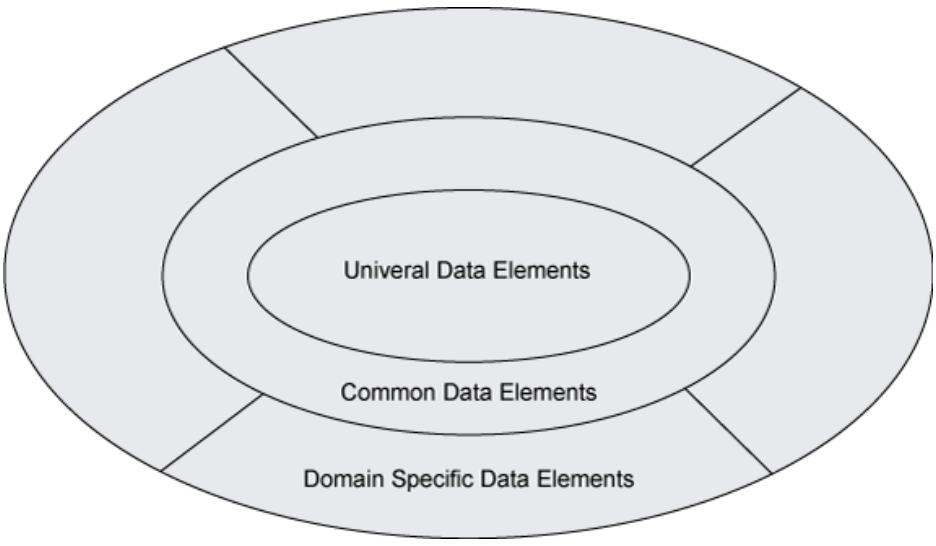
The NIEM is a federal XML-centric metadata standard created for the precise exchange of documents. Although the scope of many of the NIEM sub-domains concerns national security issues, the NIEM is successfully implemented in other domains, such as K-12 education and property taxation. The NIEM contains a general "upper ontology" that is applicable to many other domains that deal with concepts such as activities, documents, organizations, regions (GIS), and persons.

There are many benefits for beginning a Web application with a controlled vocabulary or metadata registry such as the NIEM. Metadata registries contain useful information that can be used to create a consistent set of XML schemas and forms. Using a metadata registry also forces users to declare early in the application lifecycle exactly what data elements they will transmit between organizations.

NIEM components

The NIEM is a metadata registry consisting two concentric rings of shared data elements with radial extensions for different domains.

Figure 1. NIEM ring structure



The core of the NIEM represents the "universal" data elements: the most common data exchanged between federal, state, and county organizations. Examples of universal data elements include a personâ##s name, address, contact information, document metadata, organization identifiers, and a large number of activities.

Around the universal data elements are "common" data elements. Although used by more than one organization, they are not found as frequently as universal data elements. Common data elements include items such as codes for a personâ##s eye color or codes classifying types of jewelry.

Surrounding the common data elements are domain-specific data elements. Although most NIEM domains are tied to the U.S. Department of Homeland Security, custom domains can and do work well with the NIEM universal and core domains. Figure 2 shows a search preferences screen used by the NIEM tools that allow sub-domains to be included or excluded in data element searches.

Figure 2. NIEM domain search

NIEM process

The core process in using the NIEM involves creating XML exchange document packages. These packages include an XML Schema file (.xsd) that specifies the constraints of an exchange. These constraints include:

- a listing of elements involved in each exchange (wantlist)
- the order and grouping of these elements
- the specifications of data elements that are required and optional for a valid exchange (the cardinality)

A NIEM constraint file imports other XML Schemas called NIEM sub-schemas. These sub-schemas are created using the NIEM subset generation tools. You can use the "shopping cart" metaphor to describe how non-programmers use the NIEM tools to shop for the metadata elements to place in their forms. In addition, just like shopping in a grocery store, you do not need to know the business rules of 10,000 SKUs to purchase 10 items. Analogously, a typical form may need only 20 data elements from the NIEM model, so importing the 4000+ types and classes is not efficient.

Sub-schemas are subsets of the NIEM schema but remain consistent with NIEM structures. Each imported sub-schema uses its own namespace and each data element in a sub-schema contains its own data element definition. NIEM sub-schemas easily combine with state and industry sub-schemas because they reside in their own namespaces. The NIEM tools allow you to save your "shopping list" data selection in an XML file called a "wantlist." A wantlist can be saved and re-loaded back into the NIEM tools as needed for future sessions.

The creation of separate constraint and imported data element definition files leverages the Separation of Concerns design pattern. The constraints are unique to that exchange package but the leaf-level data element definitions are universal to all documents that use NIEM standards. All exchange documents that import a NIEM generated sub-schema use the same meaning or semantics for that data element. This separation of concerns is a central design pattern in forms generation that does not involve IT staff and will become more common as semantic Web technologies continue to evolve.

NIEM naming and design rules

NIEM constraint files follow XML data element naming conventions that are consistent with other federal and international standards, such as the ISO/IEC 11179 metadata registry standards. These conventions allow general XML transforms to transform the NIEM documents into other structures.

These conventions give us five critical bits of information associated with each leaf-level data element in an XML Schema:

- **Namespace** - such as http://niem.gov/niem/universal/1.0
- **Concept** (object-class) - such as "Person"
- **Property** â## usually a short word or words that describes the property itself
- **Representation Term** - such as Code, Date, Indicator, Name, or Text
- **Data Element Definition** â## a brief text description of the element in text

When a user selects a data element to be included in a constraint schema, a data element is added to an XML Schema source file. For example, if you select **PersonGivenName**, the following code (and elements it depends upon) are added to the NIEM universal subschema:

Listing 1. XML code added when the PersonGiveName data element is added to a NIEM subschema.

```
<xsd:element name="PersonGivenName" type="u:PersonNameTextType" nillable="true">
      <xsd:annotation>
         <xsd:documentation>A first name of a person.</xsd:documentation>
      </xsd:annotation>
 </xsd:element>
```

This element is then "referenced" (using the `ref` attribute) by the main constraint schema. A person record that has required values for first name, last name, and e-mail would be represented as follows in a NIEM constraint XML schema file:

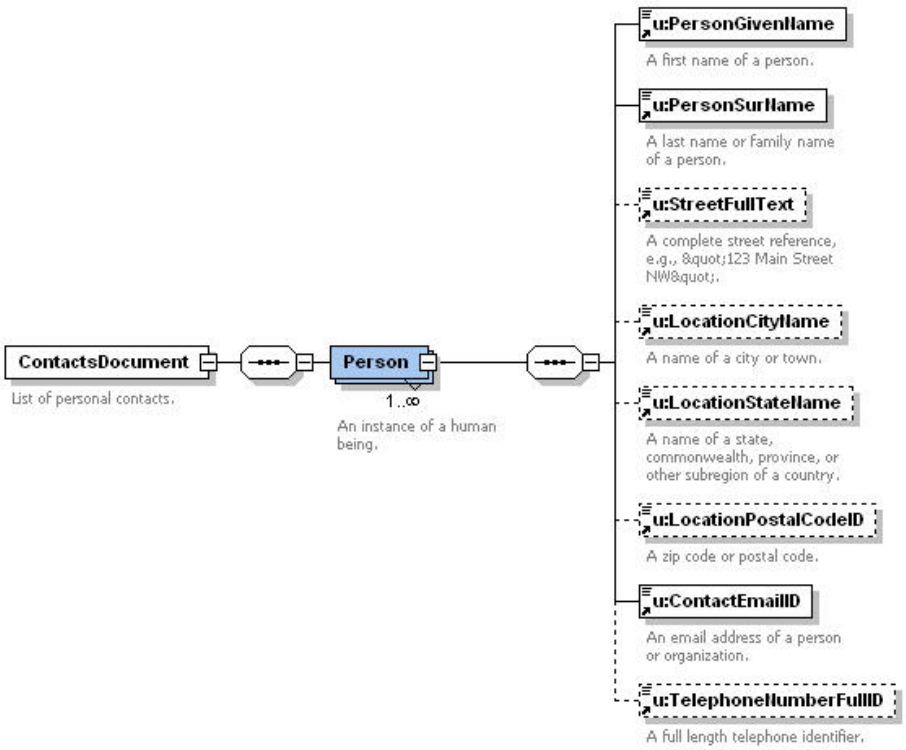
Listing 2. Person element in NIEM constraint XML Schema

```
<xs:element name="Person" maxOccurs="unbounded">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="u:PersonGivenName"/>
         <xs:element ref="u:PersonSurName"/>
         <xs:element ref="u:StreetFullText" minOccurs="0"/>
         <xs:element ref="u:LocationCityName" minOccurs="0"/>
         <xs:element ref="u:LocationStateName" minOccurs="0"/>
         <xs:element ref="u:LocationPostalCodeID" minOccurs="0"/>
         <xs:element ref="u:ContactEmailID"/>
         <xs:element ref="u:TelephoneNumberFullID" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Note that when the attribute `minOccurs=â##0â##` is located in an element, the field is optional. When `minOccurs` is not present, the default of `minOccurs=â##1â##` is assumed. This implies the element is required for the person element to be valid.

Although most programmers feel comfortable reading the source code for XML Schema files, most non-programmers prefer using a graphical representation of an XML Schema. Figure 3 is a diagram from the XMLSpy XML editing program:

Figure 3. XMLSpy schema diagram



XMLSpy and other graphical schema capture tools are excellent as requirements capturing tool for some of the following reasons:

- The NIEM-generated annotations of referenced elements (the definitions) are clearly visible under each data element. Since they are imported, they can be set to read-only and not changeable since their definitions are part of the federal standard.
- Novice users can easily drag elements around the diagram to change the order of the data elements in the schema.
- Right-click menus are used to easily guide the user in designating elements as required or optional. Dashed lines are used for optional data elements and solid lines are used for required data elements.
- All of the operations can be performed for a group of people using an interactive whiteboard such as a SmartBoard. This process is critical for the empowerment of stakeholders who desire to design and change constraints on the fly. In addition, it will demystify the process and provide focus to data stewardship.
- All complex elements (elements without namespace prefixes in the diagram above) can be easily annotated by a simple right-click over the elements.

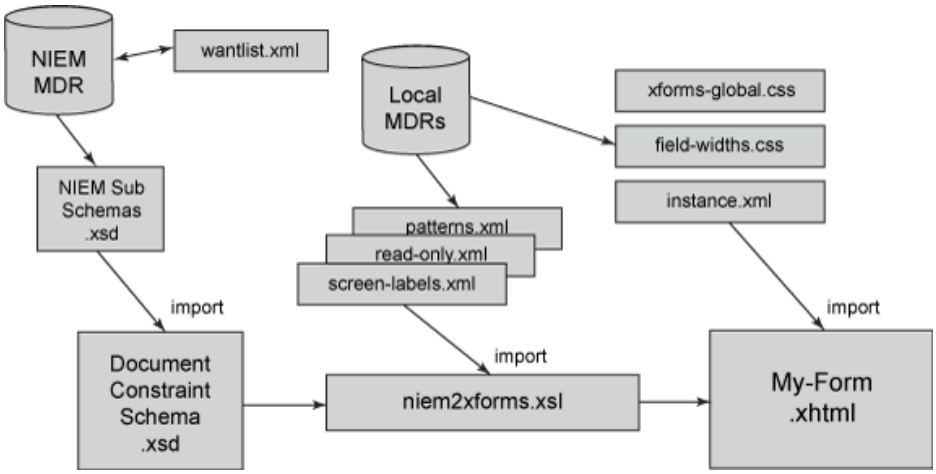
Because of the richness of each data element, XML transformations of this XML Schema can rely on a rich set of metadata in each XML Schema to build precise forms. In addition, a look-up table strategy can be used to add additional metadata to each data element when the NIEM metadata is not sufficient. I prefer this strategy over the technique of adding additional data to XML Schema appinfo structures, since external forms can easily maintain this information.

People unfamiliar with the NIEM sometimes consider NIEM tag names to be too long and the required use of namespaces burdensome for the novice XML developer. However, the automatic generation of Web forms would be much more difficult without this structure.

Transforming NIEM constraint schemas

Now that you have an idea of the structure of the NIEM, you are ready to begin to transform the constraint XML Schemas into XForms. Figure 4 describes the data flow used to create an XForms application by transforming the NIEM constraint document directly into the XForms document.

Figure 4. NIEM to XForms data flow diagram



The actual NIEM-generated and imported sub-schema files are not actually used in the constraint to XForms transformation process. Other metadata (such as screen labels) can be extracted

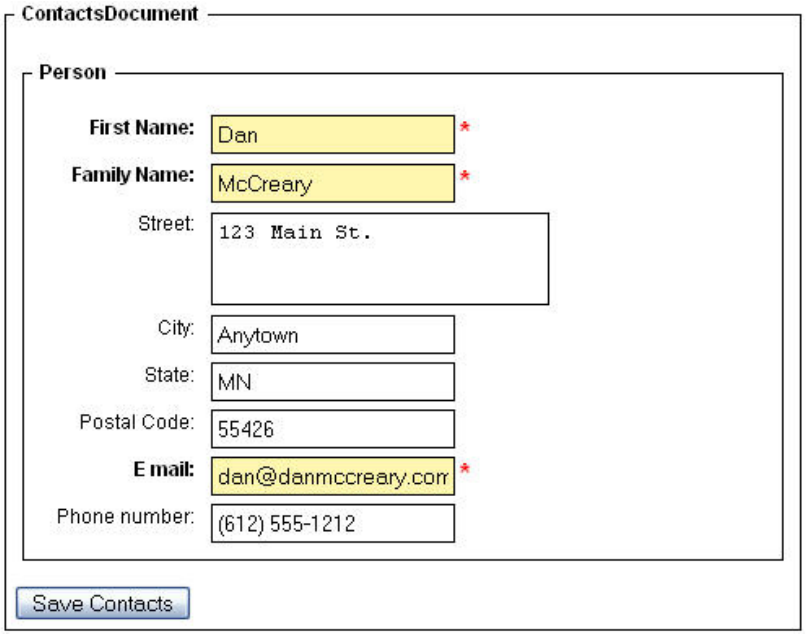
from local metadata registries and imported into the niem2xforms transform. After these files are imported as XSL variables, they can be used in the forms with a lookup-table strategy (see Listing 3).

Listing 3. XForms input controls for person

```
<xf:group ref="Person">
   <xf:label class="group-label">Person</xf:label>
   <xf:input ref="u:PersonGivenName">
      <xf:label>First Name: </xf:label>
   </xf:input>
   <xf:input ref="u:PersonSurName">
        <xf:label>Family Name: </xf:label>
    </xf:input>
    <xf:textarea ref="u:StreetFullText">
        <xf:label>Street: </xf:label>
    </xf:textarea>
    <xf:input ref="u:LocationCityName">
        <xf:label>City: </xf:label>
    </xf:input>
    <xf:input ref="u:LocationStateName">
        <xf:label>State: </xf:label>
    </xf:input>
    <xf:input ref="u:LocationPostalCodeID">
        <xf:label>Postal Code: </xf:label>
    </xf:input>
    <xf:input ref="u:ContactEmailID">
        <xf:label>E mail: </xf:label>
    </xf:input>
    <xf:input ref="u:TelephoneNumberFullID">
        <xf:label>Phone number: </xf:label>
    </xf:input>
</xf:group>
```

Here is a visual representation of this form when opened using the Firefox browser with the XForms 0.8 add-on:

Figure 5. Initial rendering of XForms file using Firefox 0.8 add-on



Mapping representation terms to controls

Because each NIEM data element uses ISO/IEC 11179 Representation Terms as the suffix for the data element name, we can use this information to map leaf-level elements directly to a specific type of control and HTML class for styling. Table 1 shows the most common NIEM Representation Terms used and how the transform automatically maps these terms into XForms controls.

Table 1. Mapping NIEM Representation Terms into XForms controls

Representation Term	Usage	XForms Control
Amount	Monetary value with units of currency.	input class="amount"
Code	An enumerated list of all allowable values. Each enumerated value is a string that for brevity represents a specific meaning.	select1 or select
Count	Non-monetary numeric value or count with units.	input class="count"
Date	An ISO 8601 date usually in the format YYYY-MM-DD	input class="date"
Identifier or ID	A language-independent label, sign or token used to establish identity of, and uniquely distinguish one instance of an object within an identification scheme.	input class="id"
Indicator	Boolean, exactly two mutually exclusive values (true or false). A precise definition must be given for the meaning of a true value.	input class="boolean"
Measure	Numeric value determined by measurement with units. Typically used with items such as height or weight. if the unit of measure is not clear it should be specified.	input class="measure"
Name	A textual label used as identification of an object. A name is usually meaningful in some language, and is the primary means of identification of objects for humans. Unlike an identifier, a name is not necessarily unique.	input class="name"
Number	Assigned or determined by calculation.	input class="number"
Text	Character string generally in the form of words.	textfield class="name"
Value	A type of Numeric.	input class="value"
Percent or Rate	A type of Numeric that traditionally is the results of a ratio calculation that ranges from values of 0 to 1 for values of 0% to 100%.	input class="percent"

Representation Terms are not unique to the NIEM data model. Representation Terms are used as a core classification scheme by many metadata registries that follow ISO/IEC 11179 metadata registry standards. See the resources section for more on Representation Terms.

Using XSLT

Using XML Transforms (XSLT) is a logical choice to transform documents when the source and destination are both well-formed XML files. XSLT can concisely and efficiently manipulate XML Schema files to perform a number of tasks, including creating XForms, generating instance documents, documenting data structures and interfaces, and controlling a variety of user interface elements. XSLT 2.0 also provides many features that make these transforms modular and easier to maintain.

There are several transformation strategies used to convert XML Schemas into other XML formats:

- Find the data elements that you are looking for in an XML Schema using template matches
- Generating the output elements in the correct order
- Mapping data types to the correct XForms controls
- Use the metadata in the constraint XML Schemas to "look up" related data from a metadata registry

This article provides a basic XSLT 2.0 transform used to convert an NIEM constraint XML Schema to an XForms application. This transform will allow non-programmers to create basic forms directly from the XML Schemas. It is intended as a starting point for developers but for simplicity's sake, this version does not include advanced features such as management of complex groups, repeated fields, and the placement of insert and delete triggers.

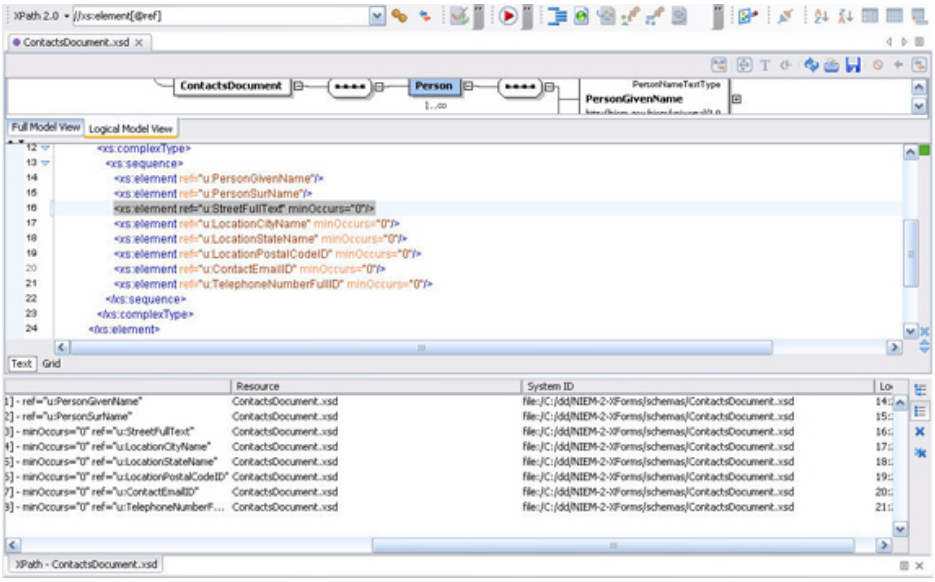
The remainder of this article describes how to modify these transforms to suit your specific business requirements.

Basic XPath expressions for traversing XML Schemas

To use and extend these transforms, one requires an understanding of how the underlying XML transformation process works. A basic understanding of XPath expressions, how templates are matched, and XSLT recursive algorithms will result in the creation of small, easy, and maintainable transforms.

The best way to learn how XPath is used to transform XML Schemas is to use an XPath evaluation tool. Most XML development tools include XPath evaluation. An example of the oXygen XML editor is listed in Figure 6.

Figure 6. Using the oXygen XML editor to learn XPath



To use the XPath tool, enter the XPath expression into the text field at the top of the screen. The bottom of the screen shows the "matches" to the XPath query.

To begin, let's first look at a few sample XPath expressions that transform XML Schemas. All XML Schema elements begin with the "xs" prefix to indicate they are in the XML Schema namespace. To use these you should use your favorite XML editor (XMLSpy, Stylus Studio, oXygen, etc.) to use the XPath evaluation functions. Open the sample supplied ContactsDocument.xsd file or one of your own and enter the following expressions.

```
//xs:element
```

Matches all elements in an XML Schema file. This returns a list of all the elements in an XML Schema regardless of where they are.

```
//xs:element[@ref]
```

Matches all elements that have a ref attribute. Note that this does not return the actual referenced element, it only returns the element nodes that have a ref attribute.

```
//xs:sequence | //xs:choice | //xs:all
```

Matches any model (any one of sequence, choice or all)

```
//xs:element[@type='xs:string'] | //xs:restriction[@base='xs:string']
```

Matches all elements that are of type string or restrictions of a base type string

Using lookup tables to add metadata

One of the challenges in using a metadata registry such as the NIEM is that it does not have the ability to store organization-specific or user-specific metadata. Examples of this additional data might include:

- Screen labels
- Field widths for specific data types, such as four digits for a year
- Additional validation patterns for data entry
- Help/hint text for data entry

The example program has examples of the first two of these. The first is done using a lookup table and the second is done by generating a CSS file that has the widths for each data element used.

There are two primary strategies for adding this metadata to a set of forms. The first involves adding it to `appinfo` tags in the constraint schema. There are two drawbacks to this design. The first is that GUI XML Schema editing tools do not have an easy way for non-programmers to modify this data and to validate this data. The second is that changing all data element labels on a single element in a family of XML Schemas requires modifying many XML Schema constraint files.

An alternative approach is to use a lookup table strategy that relies on code tables. These code tables are extracted from a central metadata registry of data common and put into a format of XSL "variables" that are imported into the XSLT 2.0 file. The samples included with this article demonstrate this technique.

For example, the format of an XML file of screen labels lookup table might be the following:

Listing 4. XForms input controls for person

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:variable name="lookup-screen-labels">
      <data>
         <item>
            <from>PersonGivenName</from>
            <to>First Name</to>
         </item>
         <item>
            <from>PersonMiddleName</from>
            <to>Middle Name</to>
         </item>
...etc.
```

Once these code tables are created, they are very easy to use in a transform. The XSLT 2.0 transform to use these values would contain the following lines:

Listing 5. XForms input controls for person

```
   <xsl:import href="screen-labels.xml"/>
â#¦
   <xsl:variable name="screen-label"
      select="$lookup-screen-labels/data/item[from=substring-after($leaf-name,':')]/to"/>
```

In Listing 5, the variable screen-label for the `from` data element is set to be the `to` value in the lookup table. In this case you only use the characters after the colon. This is an important rule

since NIEM conformant documents use namespaces and qualified data elements for all leaf-data elements.

Using CSS for form attributes

In addition to using lookup tables to extract additional metadata, other techniques can be used to make the form consistent with other user interface development standards. The CSS file can also be used to store the values of screen widths. The field widths of each element can then be expressed in terms of either the letters â##xâ## (ex) or the letter â##Mâ## (em). The following CSS file can be generated from a metadata registry and imported into the form.

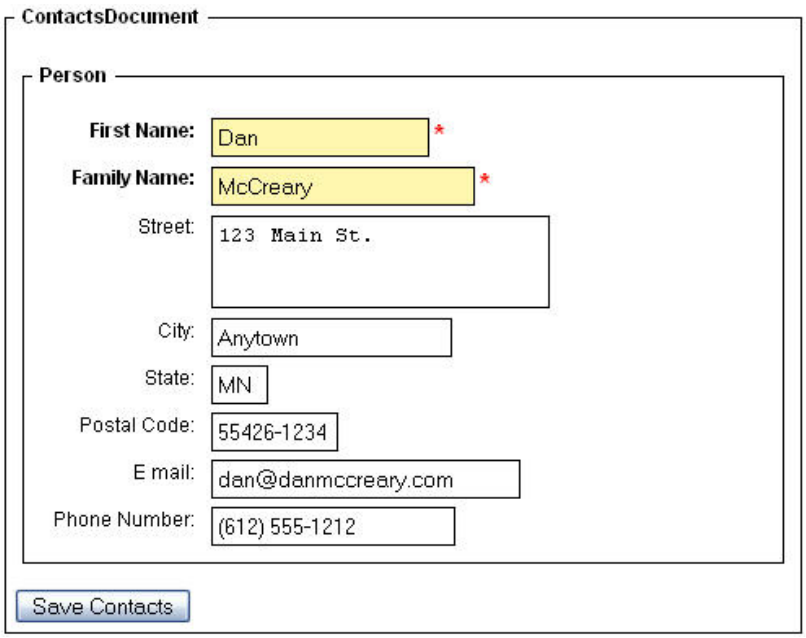
Listing 6. Using CSS to control field widths

```
@namespace xf url("http://www.w3.org/2002/xforms");

.ContactEmailID .xf-value {width: 26ex}
.PersonGivenName .xf-value {width: 18ex}
.PersonSurName .xf-value {width: 22ex}
.LocationCityName .xf-value {width: 20ex}
.LocationStateName .xf-value {width: 2em}
.LocationStateCode .xf-value {width: 2em}
.LocationPostalCodeID .xf-value {width: 10ex}
.StreetFullText .xf-value {width: 40em}
```

Listing 6 shows how CSS-3 can be used to conditionally set the width inside different elements that each used a different class attribute in the XForms input. The Firefox extension automatically adds the `.xf-value` class for all input field values. The resulting output has variable-width field, as show in Figure 7.

Figure 7. Using CSS to control field attributes



Importing selection lists from shared resource files

When developing a large family of forms, a database of shared selection lists is much easier to maintain. These selection lists can be stored in external XML â##code tablesâ## and imported

directly to an instance in the XForms model. All forms can import these resources in a consistent manner and all forms will get the updated code tables as the codes are changed. The XSL transform must insert the code into the model section of the XForms file, as described in Listing 7.

Listing 7. Importing shared code tables

```
<xf:model>
â#¦
<xf:instance xmlns="" id="PersonSexCode"
src="../resources/code-tables/PropertyTypeCode.xml" />
```

The transform adds the code in Listing 8 into the body of the form whenever an element that ends in the string â##Codeâ## in seen in the input constraint file.

Listing 8. Importing shared code tables

```
<xf:select1 ref="mn:PersnSexCode">
 <xf:label>Property Type Code:</xf:label>
 <xf:itemset model="code-tables"
          nodeset="instance('PersonSexCode')/EnumeratedValues/Item">
  <xf:label ref="Label" />
  <xf:value ref="Value" />
 </xf:itemset>
</xf:select1>
```

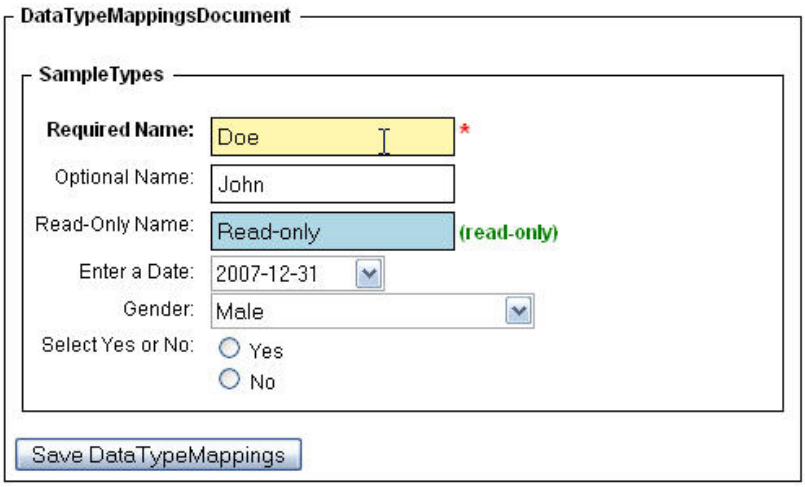
The nodeset attribute of itemset element automatically adds to the list specified in the external file to the selection list. The selection list will then display the values â##Female,â## â##Male,â## and â##Unknown.â## The cardinality of the input schema can determine if multiple values can be selected.

Modifying element displays based on data types

Just as the transform can use the â##Codeâ## suffix above, the transform can also use other data element suffixes to insert different XForms controls into the output based on the datatype of the field. For example, any data element that has a â##Dateâ## representation term can automatically insert a calendar-selector in the form. Similarly, any boolean datatypes can be converted into checkbox controls.

Figure 8 shows how the transform can work together with a CSS file to create various form presentations based on the values in the XML Schema constraint file and a local metadata registry. The example included allows dates to be selected from a calendar date selector input control.

Figure 8. Mapping data types to controls



Using XBL to extend XForms' controllers' behavior

Although many NIEM Representation Terms map directly to XForms controls, some XForms controls are not rich enough to format complex data types such as currency. Alone, CSS is not powerful enough to perform functions such as adding commas to currency when they are displayed on the screen and remove commas when they are stored in instances within the model. Some Web browsers such as Firefox now support a proposed W3C standard called XBL for XML binding language. This allows the developer to associate a small amount of JavaScript with a class for formatting and lower the burden of XML Schema to XForms transformation.

Running the sample NIEM to XForms transformation

This article includes a zip file (see the download) that has two examples of transforming NIEM XML Constraint XML Schemas directly into an XForms application. It also includes a sample Apache Ant build file and an XMLSpy project file that can execute these transformations with a few mouse clicks. A screen image of the Project file is listed in the figure below. See the README.txt file for further details about the demonstration files.

Note that the transform is an XSLT 2.0 transform and has been tested with XMLSpy and Saxon. The forms were tested with Firefox 2.0.0.4 with the XForms 0.8 add-on.

Conclusion

In this article, you saw several model-driven development techniques used to transform a XML Schema constraint file directly into a working XForms application. The transform itself is relatively short (less than 230 lines), but can be quickly customized to meet different Web form development needs.

By using NIEM XML Schema structure, naming conventions, and additional metadata, the transformation task is much easier to extend. Although the example code included in this article will create working forms, its intent is a starting point to enable non-programmers to create

working XForms applications. A software developer willing to become familiar with and modify the transformation can facilitate the extension of the transform to meet specific business requirements.

This transform is just one of the first steps an IT department can adopt to empower non-programmers to create precise specifications that automatically generate correct Web forms. This process and many similar processes like it are part of the declarative revolution that has great potential to lower overall IT development costs and empowers a much larger audience to play a direct role in Web development.

Downloadable resources

Description	Name	Size
Sample xslt programs	niem-2-xforms.zip	29KB