Extend enumerated lists in XML schema

Explore options for your extension solution

W. Paul Kiel (paul@xmlhelpline.com), President, xmlHelpline.com Consulting

Summary: The addition of new values to a list is a common and necessary requirement. Schema designers often seek to build into the architecture a means to permit additional values that were unknown at design time. How can schema designers create an enumerated value list that is extensible and easy to implement? Discover several approaches used to achieve this goal.

Date: 23 Sep 2008
Level: Intermediate
PDF: A4 and Letter (50KB | 13 pages)Get Adobe® Reader®
Also available in: Chinese Japanese

Schema designers and implementers need a way to extend existing enumerated lists in XML schema. Unfortunately, the XML Schema specification does not allow for extensibility in the creation of these lists (see Resources). Values chosen at design time are fixed and are all that's available. Despite this limitation, people use various workarounds to enable the extension of lists. This functionality is a frequent request from my clients, many of whom work with existing schemas that cannot be changed. They want to add new functionality while maintaining backward compatibility. In this article, you see how schema designers overcome barriers to enable this functionality.

An *enumerated list* is a set of specified values for a particular data point. For example, you might view a country code as a fixed list of values, including *DE* (Germany), *US* (United States), and *JP* (Japan). Given this value set, what happens when a new country is recognized, such as *TL* (East Timor) or *BA* (Bosnia and Herzegovina)? Anyone who uses the previous list of names will have to change the implementation to accommodate the new values.

When you model data with XML schema, enumerated values are listed explicitly. So, a list of country codes includes each one in turn. Recognizing that new values to a list are common and must be accommodated, schema designers have long sought a way to extend enumerated lists—in effect, to build into the design a way to permit additional values that were not known at design time.

Creating extensible enumerated lists

In finding solutions to this problem, four key criteria affect which approaches are available to extend lists:

- First is a need to extend a list after design time. Whether it's setting up a new trading partner quickly or time-critical new data fields, last-minute extensions are a real-world necessity.
- Second, the ability to validate the values in the parser is key to ease of implementation.
- Third, the ability to parse and validate in a single pass is critical. This would rule out solutions like Genericode where validation occurs in a separate pass and parser. Adding new technology requirements can be too costly or time-consuming in some settings.
- Finally, a solution must be backward compatible with the original schema. A change to a list that is not compatible is not an extension.

Some folks say that you should not extend enumerated lists at all. Data modelers might argue that if you want the model to have more data, grow the model, then create schemas as a by-product—in effect, create a bigger model and restrict down as needed. This makes sense if you have control over the original schemas and data model, and it might be the ideal approach. But if you need to extend after design time in practical reality, such an approach might not be possible.

Still others say that the key to extending enumerations is to work outside the realm of the XML schema validating parser. The Genericode effort (see Resources) recommend the validation of enumerated lists in a second layer and outside the initial XML schema parser validation process. The theory is good, and this approach might be more widely used over time. However, if you want to do it in one parser pass, this solution won't work. In some cases, a second validation pass is not possible.

Of course, you can always create a new element with a new list. But this isn't backward compatible with the original schema. The ideal is to accommodate an extensible list while you maintain backward compatibility (see Resources).

For the purposes of this article, the assumptions come from my experience with clients—namely, a desire to extend an existing enumerated list with additional values. I also assumed that it is to be done with the XML schema parser and validated along with everything else in one step.

Requirements for extending enumerated lists

This extension example has four requirements:

- Allow for extensible enumerated values after design time.
- Validate enumerations with the parser.
- Validate enumerations in a single pass.
- Maintain backward compatibility with the original schema.

Take, for example, a team that works with a domain industry consortia enumerated list (or any preexisting list) and adapts schema components for its use cases. The preexisting schema offers a `MaritalStatus` component with an enumerated list of values, as in Listing 1.

Listing 1. Marital Status enumerated list

```
<xsd:simpleType name="MaritalStatusEnumType">
    <xsd:restriction base="xsd:normalizedString">
        <xsd:enumeration value="Divorced"/>
        <xsd:enumeration value="Married"/>
        <xsd:enumeration value="NeverMarried"/>
        <xsd:enumeration value="Separated"/>
        <xsd:enumeration value="SignificantOther"/>
        <xsd:enumeration value="Widowed"/>
        <xsd:enumeration value="Unknown"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:element name="MaritalStatus" type="MaritalStatusEnumType"/>
```

Suppose a company wants to use these values but also needs to support another value with one of its key trading partners. This value, `CivilUnion`, is an extension value, and the company realizes that it's not part of the original schema. But semantically, it makes sense to use that existing element—`MaritalStatus`—for the same purpose. How can the company do this?

Solution 1: Edit the original schema to include the new enumerated values

Editing the original schema to include the new enumerated values is, of course, the most direct approach. Keep a local copy of the schemas, and edit them to support the enumerated values your enterprise uses.

- **Advantage:** Easy to do
- **Disadvantages:**
  - Requires editing of original schemas, which themselves can change over time and be outside your control. If you extend a preexisting list, then the originator (trading partner, consortia, and the like) might issue a new revision of the list. You need to propagate the edits over each new version.
  - Hand-editing schemas can lead to inadvertent editorial errors.

If you can't (or don't want to) edit the original schemas, you need an alternate method.

Solution 2: Create a new enumerated list, and join the lists

The second option is to create a new enumerated list and join it to the original enumerated list. Listing 1 showed the original Marital Status list. Listing 2 shows the newly created enumerated list.

Listing 2. New Marital Status enumerated list

```
<xsd:simpleType name="MyExtMaritalStatusEnumType">
    <xsd:restriction base="xsd:normalizedString">
        <xsd:enumeration value="CivilUnion"/>
    </xsd:restriction>
</xsd:simpleType>
```

You combine the lists using the `<xsd:union>` tag, as in Listing 3.

Listing 3. Union of lists

```
<xsd:simpleType name="MaritalStatusType_Union">
     <xsd:union memberTypes="MyExtMaritalStatusEnumType MaritalStatusEnumType"/>
</xsd:simpleType>

<xsd:element name="MaritalStatus" type="MaritalStatusType_Union"/>
```

This solution still requires an edit to the schema—namely, changing the `MaritalStatus` element to be of type `MaritalStatusType_Union` instead of `MaritalStatusType`. A little less intrusive, but still some minor hand editing.

- **Advantage:** The original enumeration list is not touched.
- **Disadvantages:**
  - All values must be known at design time, preventing late binding solutions.
  - Requires `<xsd:union>` tag support, which is sometimes not implemented in tools.

Solution 3: Create a pattern, and combine it with the original enumerated type

Now switch to a use case that involves the demographic data of a person's eye color. Listing 4 shows this list.

Listing 4. Person Eye Color enumerated list

```
<xsd:simpleType name="PersonEyeColorType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Black"/>
        <xsd:enumeration value="Hazel"/>
        <xsd:enumeration value="Gray"/>
        <xsd:enumeration value="Brown"/>
        <xsd:enumeration value="Violet"/>
        <xsd:enumeration value="Green"/>
        <xsd:enumeration value="Blue"/>
        <xsd:enumeration value="Maroon"/>
```

```
        <xsd:enumeration value="Pink"/>
        <xsd:enumeration value="Dichromatic"/>
        <xsd:enumeration value="Unknown"/>
    </xsd:restriction>
</xsd:simpleType>
```

Next, create a pattern (a regular expression) that can accommodate new values. This pattern is simply any string preceded by `x:`. The `x:` is the delineator between standard enumerations and extensions. Listing 5 shows this pattern.

Listing 5. Regular expression for extension use

```
<xsd:simpleType name="StringPatternType">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="x:\S.*"/>
    </xsd:restriction>
</xsd:simpleType>
```

Finally, you combine the list and the pattern using the `<xsd:union>` tag, as in Listing 6.

Listing 6. Union of enumerated list and extension pattern

```
<xsd:simpleType name="MyExtPersonEyeColorType">
    <xsd:union memberTypes="PersonEyeColorType StringPatternType"/>
</xsd:simpleType>

<xsd:element name="PersonEyeColor" type="MyExtPersonEyeColorType"/>
```

The same node carries standard and extended values. Each is easily separated, and each can be validated by the parser, as in Listing 7.

Listing 7. Sample XML instances

```
<PersonEyeColor>Black</PersonEyeColor>
<PersonEyeColor>x:Teal</PersonEyeColor>
```

- **Advantages:**
  - The same element is used for all data.
  - Validation of the base enumeration list is done by the parser.
  - There is clear separation of extended values.
  - This solution allows for late binding of new values.
- **Disadvantages:**
  - You must parse the content of the element to determine whether it's extended.
  - The schema parser must have support for regular expression facets.
  - `<xsd:union>` tag support is required.

Solution 4: Use a separate field for extensions

In this solution, the enumeration field is not changed. However, you design an extension field into the schema to accommodate additional values. In this case, the initial list is dependent types (the relationship between an employment beneficiary and a dependent person), as in Listing 8.

Listing 8. Dependent Relationship enumeration list

```
<xsd:simpleType name="DependentRelationshipEnumType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="AdoptedChild"/>
        <xsd:enumeration value="Brother"/>
        <xsd:enumeration value="Child"/>
        <xsd:enumeration value="ExSpouse"/>
        <xsd:enumeration value="Father"/>
        <xsd:enumeration value="Granddaughter"/>
        <xsd:enumeration value="Grandson"/>
        <xsd:enumeration value="Grandfather"/>
        <xsd:enumeration value="Grandmother"/>
        <xsd:enumeration value="LifePartner"/>
        <xsd:enumeration value="Mother"/>
        <xsd:enumeration value="Sister"/>
        <xsd:enumeration value="Spouse"/>
        <xsd:enumeration value="Extension"/>
    </xsd:restriction>
</xsd:simpleType>
```

An additional attribute—`extension`—is needed that can accommodate new values. Listing 9 shows this attribute.

Listing 9. Dependent Relationship extension attribute

```
<xsd:complexType name="DependentRelationshipType">
    <xsd:simpleContent>
        <xsd:extension base="DependentRelationshipEnumType">
            <xsd:attribute name="extension" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>

<xsd:element name="DependentRelationship" type="DependentRelationshipType"/>
```

Listing 10 shows some XML instances reflecting the extension.

Listing 10. Sample XML instances

```
<DependentRelationship>Child</DependentRelationship>
<DependentRelationship extension="MyNewRelationship">Extension</DependentRelationship>
```

- **Advantages:**
  - No editing of the original schema is necessary.
  - This solution accommodates late binding of new values.
  - The `extension` method is explicitly designed into the original schema.
- **Disadvantages:**
  - The `extension` method must be designed into each enumerated list at design time.
  - Enumerated values must occur in elements, not attributes.

Solution 5: Documentation-based approach -- union with string

Note: Solutions 5 and 6 violate the *validation in one pass* requirement. However, I include them here because they are approaches used in many real contexts.

In the fifth solution, you use an enumerated list that combines with a string using the `<xsd:union>` tag. This solution, in effect, gives the receiving system a hint as to which values are standard (including casing and spelling) but in fact allows any value in the string field. So, the parser does not validate the values. Instead, they are validated in a second pass or in the application receiving the data. This is the strategy used in some XML consortia, for example.

Listing 11 shows an enumerated list combined through `<xsd:union>` with an `<xsd:string>`. Because any value can be a string, the enumerations not validated. They are suggested standard values.

Listing 11. DayOfWeek enumerated list combined with string

```
<xsd:simpleType name="DayOfWeekEnumType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Sunday"/>
        <xsd:enumeration value="Monday"/>
        <xsd:enumeration value="Tuesday"/>
        <xsd:enumeration value="Wednesday"/>
        <xsd:enumeration value="Thursday"/>
        <xsd:enumeration value="Friday"/>
        <xsd:enumeration value="Saturday"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:element name="DayOfWeek" type="DayOfWeekEnumType"/>

<xsd:simpleType name="ExtendedDayOfWeekType">
        <xsd:union memberTypes="DayOfWeekEnumType xsd:string"/>
</xsd:simpleType>
<xsd:element name="DayOfWeek_solution5" type="ExtendedDayOfWeekType"/>
```

- **Advantage:** Infinite extension values can be added, even in late binding.
- **Disadvantages:**
  - Enumerated values are not validated by the parser but in some second step.
  - `<xsd:union>` tag support is required.

Solution 6: Documentation-based approach -- using <xsd:annotation>

To employ this approach, you put actual enumeration values in the `<xsd:documentation>` tag while you leave the data field as a simple string. Listing 12 shows the enumerated values.

Listing 12. Enumerated values in the <xsd:documentation> tag

```
<xsd:element name="DayOfWeek" type="xsd:string">
    <xsd:annotation>
        <xsd:documentation>
            <!-- suggested enumerations -->
            <xsd:enumeration value="Sunday"/>
            <xsd:enumeration value="Monday"/>
            <xsd:enumeration value="Tuesday"/>
            <xsd:enumeration value="Wednesday"/>
            <xsd:enumeration value="Thursday"/>
            <xsd:enumeration value="Friday"/>
            <xsd:enumeration value="Saturday"/>
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>
```

- **Advantages:**
  - Infinite extension values can be added, even in late binding.
  - Only the simplest XML schema features are required.
- **Disadvantage:** Enumerated values are not validated by the parser.

Approaches not considered

I left a couple of approaches out of my potential solutions for expanding enumerated lists. The following list provides a brief description of the two unused approaches:

- **Use of the `<xsd:redefine>` tag:** This feature of XML schema is generally not used and often not implemented in tools. It is frequently considered a best practice to avoid redefinition.
- **Use of the `substitutionGroup` element to swap in a union list containing all values:** Another tempting solution involves the use of substitution groups and union. Create an all-inclusive enumerated list using a union of the original list and a new list. Then, using `substitutionGroups` (or the `<xsi:type>` tag), substitute a globally scoped element. The problem with this approach is that a union is not a valid derivation for substitution, which requires that two components be derived from the same base type. Extension and restriction are both valid methods for a substitution; however, union is not a valid derivation technique according to the XML Schema specification (see Resources).

Conclusion

XML schema designers and implementers need a way to extend existing enumerated lists. Because the specification does not allow for this once you create an original list, a workaround is critical to make real-world implementation possible. Implementers can use the examples in this article to help design and extend enumerations. Each method has advantages and disadvantages, and none of them is the best practice in all cases. So, which approach should you use?

Consider these rules of thumb:

- If you feel comfortable editing the original enumerated list or schema and know all the extended enumerated values at design time, solution 1 (hand-edit the original) or solution 2 (create a new list and join it to the original) might work best.
- If you want to use the same semantic element to contain both base enumerations and extended enumerations, consider solution 3 (union with a pattern).
- If you don't mind having different fields for lists and extensions, then solution 4 (separate fields) will work.
- If you want to keep enumerated values out of the parser, consider the Genericode approach, or use solution 5 or solution 6.

These guidelines can empower schema designers with real-world, practical best practices and can help create easy-to-implement, extensible enumerated lists.

Download

Description	Name	Size	Download method
XML schema and XML instance examples	ExtendEnumeratedListsCode.zip	2KB	HTTP

Information about download methods

Resources

Learn

- Profiling XML Schema (Paul Kiel, xml.com, 2006): Read this 2006 companion to this current article. It is a continuation of the goal to empower schema designers with real-world implementation practices—in this case, examining methods to extend enumerated lists.
- W3C XML Schema specification: Read how to define the structure, content, and semantics of XML documents.
- XML schema subsets and backward compatibility: Read the blog post about a perennial request in the data integration space—the ability to create a subset schema.
- XML-DEV posting on enumerated lists: Check out the posting on this topic.
- XML schema restrictions and extensions: Read the blog post for examples of both.
- Genericode: Check out the site for news and specifications on a standard format for defining code lists, including how to derive new code lists from existing code lists.
- Reference Data and Enumerated List Implemented in XML (John Bobbitt, 2004): Read this article for another examination of enumeration extension methods.
- Restriction and extension are valid derivations for substitution: Check out the XML-DEV list posting discussing this topic.
- IBM XML certification: Find out how you can become an IBM-Certified Developer in XML and related technologies.
- XML technical library: See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- developerWorks technical events and webcasts: Stay current with technology in these sessions.
- The technology bookstore: Browse for books on these and other technical topics.
- developerWorks podcasts: Listen to interesting interviews and discussions for software developers.

Get products and technologies

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

Discuss

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

About the author

Paul Kiel is president of xmlHelpline.com, a consulting firm that specializes in enabling standards. XML schema design and best practices, data modeling, XSLT, and data integration are particular emphases. He previously held a position as Chief Architect of the HR-XML Consortium and has worked with XML technology since 1997. Paul was originally trained as a librarian and archivist and often indulges his other passion of collecting historical documentaries. You can reach Paul at paul@xmlhelpline.com.

Add comment:

Sign in or register to leave a comment.

Note: HTML elements are not supported within comments.

Notify me when a comment is added1000 characters left

Post

Total comments (1)

This was a lifesaver. Solution 4 appears to allow a standard to publish schema which contains code lists that can be customized within an enterprise without changing the standard schema or breaking backwards compatibility ... and even better, this is only true for the set of lists which are enabled.

A simple but elegant solution. Great stuff

Posted by **nvrijn** on 14 June 2011

Report abuse