

Creating Extensible Content Models

Table of Contents

- Issue
- Definition
- Introduction
- Extensibility via Type Substitution
- Extensibility via the <any> Element
- Non-determinism and the <any> element
- Best Practice

Issue

What is Best Practice for creating extensible content models?

Definition

An element has an extensible content model if in instance documents the authors can extend the contents of that element with additional elements beyond what was specified by the schema.

Introduction

```
<xsd:element name= "Book">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Title" type="string"/>
      <xsd:element name="Author" type="string"/>
      <xsd:element name="Date" type="string"/>
      <xsd:element name="ISBN" type="string"/>
      <xsd:element name="Publisher" type="string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

This schema snippet dictates that in instance documents the <Book> elements must always be comprised of exactly 5 elements <Title>, <Author>, <Date>, <ISBN>, and <Publisher>. For example:

```
<Book>
  <Title>The First and Last Freedom</Title>
  <Author>J. Krishnamurti</Author>
  <Date>1954</Date>
  <ISBN>0-06-0064831-7</ISBN>
  <Publisher>Harper & Row</Publisher>
</Book>
```

The schema specifies a fixed/static content model for the Book element. Book's content must rigidly conform to just the schema specification. Sometimes this rigidity is a good thing. Sometimes we want to give our instance documents more flexibility.

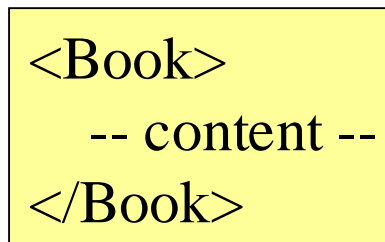
How do we design the schema so that Book's content model is extensible? Below are two methods for implementing extensible content models.

Extensibility via Type Substitution

Consider this version of the above schema, where Book's content model has been defined using a type definition:

```
<xsd:complexType name="BookType">
  <xsd:sequence>
    <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:sequence>
</xsd:complexType>
<xsd:element name="BookCatalogue">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Book" type="BookType"
        maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Recall that via the mechanism of type substitutability, the contents of <Book> can be substituted by any type that derives from BookType.



```
<Book>
  -- content --
</Book>
```

For example, if a type is created which derives from BookType:

```
<xsd:complexType name="BookTypePlusReviewer">
  <xsd:complexContent>
    <xsd:extension base="BookType" >
      <xsd:sequence>
        <xsd:element name="Reviewer" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

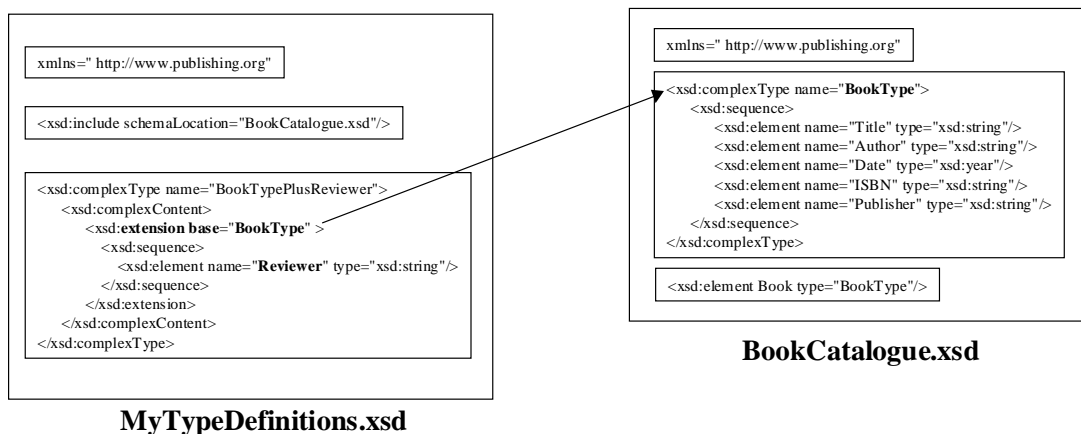
then instance documents can create a <Book> element that contains a <Reviewer> element, along with the other five elements:

```
<Book xsi:type="BookTypePlusReviewer">
  <Title>My Life and Times</Title>
  <Author>Paul McCartney</Author>
  <Date>1998</Date>
  <ISBN>94303-12021-43892</ISBN>
  <Publisher>McMillin Publishing</Publisher>
  <Reviewer>Roger Costello</Reviewer>
</Book>
```

Thus, Book's content model has been extended with a new element (Reviewer)!

In this example, BookTypePlusReviewer has been defined within the same schema as BookType. In general, however, this may not be the case. Other schemas can import/include the BookCatalogue schema and define types which derive from BookType. Thus, the contents of Book may be extended, without modifying the BookCatalogue schema, as we see on the next page:

Extend a Schema, without Touching it!



And here's what an instance document would look like:

```
xmlns="http://www.publishing.org"
```

```
xsi:schemaLocation="http://www.publishing.org  
MyTypeDefinitions.xsd"
```

```
<Book xsi:type="BookTypePlusReviewer">  
  <Title>The First and Last Freedom</Title>  
  <Author>J. Krishnamurti</Author>  
  <Date>1954</Date>  
  <ISBN>0-06-064831-7</ISBN>  
  <Publisher>Harper & Row</Publisher>  
  <Reviewer>Roger L. Costello</Reviewer>  
</Book>
```

We have type-substituted Book's content with the type specified in the new schema. Thus, we have extended BookCatalogue.xsd without touching it!

This type substitutability mechanism is a powerful extensibility mechanism. However, it suffers from two problems:

Disadvantages:

Location Restricted Extensibility: The extensibility is restricted to appending elements onto the end of the content model (after the <Publisher> element). What if we wanted to extend <Book> by adding elements to the beginning (before <Title>), or in the middle, etc? We can't do it with this mechanism.

Unexpected Extensibility: If you look at the declaration for Book:

```
<xsd:element name="Book" type="BookType"  
  maxOccurs="unbounded" />
```

and the definition for BookType:

```
<xsd:complexType name="BookType">  
  <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="xsd:string"/>  
    <xsd:element name="Publisher" type="xsd:string"/>  
  </xsd:sequence>  
</xsd:complexType>
```

it is easy to be fooled into thinking that in instance documents the <Book> elements will always contain just <Title>, <Author>, <Date>, <ISBN>, and <Publisher>. It is easy to forget that someone could extend the content model using the type substitutability mechanism. Extensibility is unexpected! Consequently, if you write a program to process BookCatalogue instance documents, you may forget to take into account the fact that a <Book> element may contain more than five children.

It would be nice if there was a way to explicitly flag places where extensibility may occur: “hey, instance documents may extend <Book> at this point, so be sure to write your code taking this possibility into account.” In addition, it would be nice if we could extend Book’s content model at locations other than just the end ... The <any> element gives us these capabilities beautifully, as is discussed in the next section.

Extensibility via the <any> Element

An <any> element may be inserted into a content model to enable instance documents to contain additional elements. Here’s an example showing an <any> element at the end of Book’s content model:

```
<xsd:element name= "Book">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Title" type="string"/>
      <xsd:element name="Author" type="string"/>
      <xsd:element name="Date" type="string"/>
      <xsd:element name="ISBN" type="string"/>
      <xsd:element name="Publisher" type="string"/>
      <xsd:any namespace="##any" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

“The content of Book is Title, Author, Date, ISBN, Publisher and then (optionally) any well-formed element. The new element may come from any namespace.”

Note the <any> element may be inserted at any point, e.g., it could be inserted at the top, in the middle, etc.

In this version of the schema it has been explicitly specified that after the <Publication> element any well-formed XML element may occur and that XML element may come from any namespace. For example, suppose that the instance document author discovers a schema, containing a declaration for a Reviewer element:

```
<xsd:element name="Reviewer">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Name">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="First" type="xsd:string"/>
            <xsd:element name="Last" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

And suppose that for an instance document author it is important that, in addition to specifying the Title, Author, Date, ISBN, and Publisher of each Book, he/she specify a Reviewer. Because the schema has been designed with extensibility in mind, the instance document author can use the Reviewer element in his/her BookCatalogue:

```
<Book>
  <Title>The First and Last Freedom</Title>
  <Author>J. Krishnamurti</Author>
  <Date>1954</Date>
  <ISBN>0-06-064831-7</ISBN>
  <Publisher>Harper & Row</Publisher>
  <rev:Reviewer>
    <rev:Name>
      <rev:Last>Costello</rev:Last>
      <rev:First>Roger</rev:First>
    </rev:Name>
  </rev:Reviewer>
</Book>
```

The instance document author has enhanced the instance document with an element that the schema designer may have never even envisioned. We have *empowered* the instance author with a great deal of flexibility in creating the instance document. Wow!

An alternate schema design is to create a BookType (as we did above) and embed the <any> element within the BookType:

```
<xsd:element name="Book">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string"/>
    <xsd:element name="Date" type="xsd:year"/>
    <xsd:element name="ISBN" type="xsd:string"/>
    <xsd:element name="Publisher" type="xsd:string"/>
    <xsd:any namespace="##any" minOccurs="0"/>
  </xsd:sequence>
</xsd:element>
```

and then declare Book of type BookType:

```
<xsd:element Book type="BookType"/>
```

However, then we are then back to the “unexpected extensibility” problem. Namely, after the <Publication> element any well-formed XML element may occur, and after that anything could be present.

There is a way to control the extensibility and still use a type. We can add a block attribute to Book:

```
<xsd:element Book type="BookType" block="#all"/>
```

The block attribute prohibits derived types from being used in Book’s content model. Thus, by this method we have created a reusable component (BookType), and yet we still have control over the extensibility.

With the <any> element we have complete control over *where*, and *how much* extensibility we want to allow. For example, suppose that we want to enable there to be at most two new elements at the top of Book’s content model. Here’s how to specify that using the <any> element:

```

<xsd:complexType name="Book">
  <xsd:sequence>
    <xsd:any namespace="##other" minOccurs="0" maxOccurs="2"/>
    <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:sequence>
</xsd:complexType>

```

Note how the <any> element has been placed at the top of the content model, and it has set maxOccurs="2". Thus, in instance documents the <Book> content will always end with <Title>, <Author>, <Date>, <ISBN>, and <Publisher>. Prior to that, two well-formed XML elements may occur.

In summary:

- We can put the <any> element specifically “where” we desire extensibility.
- If we desire extensibility at multiple locations, we can insert multiple <any> elements.
- With maxOccurs we can specify “how much” extensibility we will allow.

Non-Determinism and the <any> element

In the above BookType definition we used an <any> element at the beginning of the content model. We specified that the <any> element must come from an other namespace (i.e., it must not be an element from the targetNamespace). If, instead, we had specified namespace="##any" then we would have gotten a “non-deterministic content model” error when validating an instance document. Let’s see why.

A non-deterministic content model is one where, upon encountering an element in an instance document, it is ambiguous which path was taken in the schema document. For example. Suppose that we were to declare BookType using ##any, as follows:

```

<xsd:complexType name="Book">
  <xsd:sequence>
    <xsd:any namespace="##any" minOccurs="0" maxOccurs="2"/>
    <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:sequence>
</xsd:complexType>

```


And suppose that we have this (snippet of an) instance document:

```
<Book>
  <Title>The First and Last Freedom</Title>
  <Author>J. Krishnamurti</Author>
  <Date>1954</Date>
  <ISBN>0-06-064831-7</ISBN>
  <Publisher>Harper & Row</Publisher>
</Book>
```

Let's see what happens when a schema validator gets to the `<Title>` element in this instance document. The schema validator must determine what this Title element declaration corresponds to in the schema document. Do you see the ambiguity? There is no way to know, *without doing a look-ahead*, whether the Title element comes from the `<any>` element, or comes from the `<xsd:element name="Title" .../>` declaration. This is a non-deterministic content model: if your schema has a content model which would require a schema validator to "look-ahead" then your schema is non-deterministic. Non-deterministic schemas are not allowed.

The solution in our example is to declare that the `<any>` element must come from an other namespace, as was shown earlier. That works fine in this example where all the BookCatalogue elements come from the targetNamespace, and the `<any>` element comes from a different namespace. Suppose, however, that the BookCatalogue schema imported element declarations from other namespaces. For example:

```
<?xml version="1.0"?>
<xsd:schema ...
  xmlns:bk="http://www.books.com" ...>
  <xsd:import namespace="http://www.books.com"
    schemaLocation="Books.xsd"/>
  <xsd:complexType name="Book">
    <xsd:sequence>
      <xsd:any namespace="##other" minOccurs="0"/>
      <xsd:element ref="bk:Title"/>
      <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:sequence>
  </xsd:complexType>
</xsd:schema>
```

Now consider this instance document:

```
<Book>
  <bk:Title>The First and Last Freedom</bk:Title>
  <Author>J. Krishnamurti</Author>
  <Date>1954</Date>
  <ISBN>0-06-0064831-7</ISBN>
  <Publisher>Harper & Row</Publisher>
</Book>
```

When a schema validator encounters `bk:Title` it will try to validate it against the appropriate element declaration in the schema. But is this the Title referred to by the schema (i.e., in the `http://www.books.com` namespace), or does this Title come from using the `<any>` element? It is ambiguous, and consequently non-deterministic. Thus, this schema is also illegal.

As you can see, prohibiting non-deterministic content models makes the use of the `<any>` element quite restricted. So, what do you do when you want to enable extensibility at arbitrary locations? Answer: put in an optional `<other>` element and let its content be `<any>`. Here's how to do it:

```
<xsd:complexType name="Book">
  <xsd:sequence>
    <xsd:element name="other" minOccurs="0">
      <xsd:any namespace="##any" maxOccurs="2"/>
    </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:sequence>
</xsd:complexType>
```

Now, instance document authors have an explicit container element (`<other>`) in which to put additional elements. This isn't the most ideal solution, but it's the best that we can do given the rule that schemas may not have non-deterministic content models. Write to the XML Schema working group and tell them that you want the prohibition of non-deterministic content models revoked!

Best Practice

The `<any>` element is an enabling technology. It turns instance documents from static/rigid structures into rich, dynamic, flexible data objects. It shifts focus from the schema designer to the instance document author in terms of defining what data makes sense. It empowers instance document authors with the ability to decide what data makes sense to him/her.

As a schema designer you need to recognize your limitations. You have no way of anticipating all the varieties of data that an instance document author might need in creating an instance document. Be smart enough to know that you're not smart enough to anticipate all possible needs! Design your schemas with flexibility built-in.

Definition: an *open content* schema is one that allows instance documents to contain additional elements beyond what is declared in the schema. As we have seen, this may be achieved by using the <any> (and <anyAttribute>) element in the schema.

Sprinkling <any> and <anyAttribute> elements liberally throughout your schema will yield benefits in terms of how evolvable your schema is:

Enabling Schema Evolution using Open Content Schemas

In today's rapidly changing market static schemas will be less commonplace, as the market pushes schemas to quickly support new capabilities. For example, consider the cellphone industry. Clearly, this is a rapidly evolving market. Any schema that the cellphone community creates will soon become obsolete as hardware/software changes extend the cellphone capabilities. For the cellphone community rapid evolution of a cellphone schema is not just a nicety, the market demands it!

Suppose that the cellphone community gets together and creates a schema, *cellphone.xsd*. Imagine that every week NOKIA sends out to the various vendors an instance document (conforming to *cellphone.xsd*), detailing its current product set. Now suppose that a few months after *cellphone.xsd* is agreed upon NOKIA makes some breakthroughs in their cellphones - they create new memory, call, and display features, none of which are supported by *cellphone.xsd*. To gain a market advantage NOKIA will want to get information about these new capabilities to its vendors ASAP. Further, they will have little motivation to wait for the next meeting of the cellphone community to consider upgrades to *cellphone.xsd*. They need results NOW. How does open content help? That is described next.

Suppose that the cellphone schema is declared "open". Immediately NOKIA can extend its instance documents to incorporate data about the new features. How does this change impact the vendor applications that receive the instance documents? The answer is - not at all. In the worst case, the vendor's application will simply skip over the new elements. More likely, however, the vendors are showing the cellphone features in a list box and these new features will be automatically captured with the other features. Let's stop and think about what has been just described. Without modifying the cellphone schema and without touching the vendor's applications, information about the new NOKIA features has been instantly disseminated to the marketplace! Open content in the cellphone schema is the enabler for this rapid dissemination.

Clearly some types of instance document extensions may require modification to the vendor's applications. Recognize, however, that the vendors are free to upgrade their applications in their own time. The applications do not need to be upgraded before changes can be introduced into instance documents. At the very worst, the vendor's applications will simply skip over the extensions. And, of course, those vendors do not need to upgrade in lock-step

To wrap up this example suppose that several months later the cellphone community reconvenes to discuss enhancements to the schema. The new features that NOKIA first introduced into the marketplace are then officially added into the schema. Thus completes the cycle. Changes to the instance documents have driven the evolution of the schema.