Restricting complexTypes that have mixed content

Roger L. Costello
October 2012

complexType with mixed content (no attributes)

Here is a complexType with mixed content:

```
<xs:complexType name="C1" mixed="true">
    <xs:sequence>
        <xs:element name="a1" minOccurs="0" type="xs:string" />
    </xs:sequence>
</xs:complexType>
```

[Notice: the child element a1 is optional.]

Here is an element declared to be of that type:

```
<xs:element name="E1" type="C1" />
```

The content of element E1 can be any of these:

1. Pure text (xs:string)

2. Child element a1

3. Text and child element a1 (i.e., mixed content)

Here are examples to illustrate the possible content of E1:

1. E1 has pure text:

```
<E1>Hello World</E1>
```

2. E1 has a child element a1:

```
<E1>
    <a1>hello world</a1>
</E1>
```

3. E1 has text surrounding the child element a1:

```
<E1>This is some text
    <a1>hello world</a1>
    that surrounds the element
  </E1>
```

Restricting a complexType with mixed content

Recall that derive-by-restriction allows us to create a complexType with content that is a subset of a base type. Let C1 be the base type. It has two pieces of content that can be restricted away (i.e., removed):

(a) the optional child element a1 (i.e., the markup)
(b) the text that surrounds the child element

Let's implement each of these restrictions.

(a) This complexType removes C1's markup:

```
  <xs:complexType name="C1-sans-child-element">
    <xs:simpleContent>
      <xs:restriction base="C1">
        <xs:simpleType>
          <xs:restriction base="xs:string"/>
        </xs:simpleType>
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
```

[Notice: since its content is text, use simpleContent.]

Here I declare an element of that type:

```
<xs:element name="E1-sans-child-element" type="C1-sans-child-element" />
```

Its only legal content is string data:

```
<E1-sans-child-element>Hello World</E1-sans-child-element>
```

[Important: the string data is unconstrained—it can be of any length and can contain any characters (Chinese, Arabic, Cyrillic, etc.)]

[Important: C1's markup can be restricted away only if its markup is optional. The element a1 has minOccurs="0" and therefore can be removed.]

We can constrain the string data as shown here:

```
<xs:complexType name="C1-sans-child-element-constrained">
    <xs:simpleContent>
        <xs:restriction base="C1">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:maxLength value="10"/>
                    <xs:pattern value="[a-zA-Z0-9 ]*" />
                </xs:restriction>
            </xs:simpleType>
        </xs:restriction>
    </xs:simpleContent>
</xs:complexType>
```

The string data is now constrained to just 10 or less lower- and upper-case letters, digits and spaces.

(b) This complexType removes the text that can surround C1's child element:

```
<xs:complexType name="C1-sans-text">
    <xs:complexContent>
        <xs:restriction base="C1">
            <xs:sequence>
                <xs:element name="a1" minOccurs="0" type="xs:string" />
            </xs:sequence>
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>
```

[Notice: since its content is an element, use complexContent.]

Here I declare an element:

```
<xs:element name="E1-sans-text" type="C1-sans-text" />
```

Its only legal content is the child element a1:

```
<E1-sans-text>
    <a1>hello world</a1>
</E1-sans-text>
```

Equivalent Types

The content of C1-sans-child-element is just a string. So these two types are equivalent in terms of schema-valid data:

```
<xs:complexType name="C1-sans-child-element">
    <xs:simpleContent>
        <xs:restriction base="C1">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:restriction>
    </xs:simpleContent>
</xs:complexType>

<xs:simpleType name="C1-sans-child-element-equivalent">
    <xs:restriction base="xs:string" />
</xs:simpleType>
```

The content of C1-sans-text is just markup. So these two types are equivalent in terms of schema-valid data:

```
<xs:complexType name="C1-sans-text">
    <xs:complexContent>
        <xs:restriction base="C1">
            <xs:sequence>
                <xs:element name="a1" minOccurs="0" type="xs:string" />
            </xs:sequence>
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="C1-sans-text-equivalent">
    <xs:sequence>
        <xs:element name="a1" minOccurs="0" type="xs:string" />
    </xs:sequence>
</xs:complexType>
```

complexType with mixed content and an attribute

Here is a complexType with mixed content and an optional attribute:

```
<xs:complexType name="C2" mixed="true">
    <xs:sequence>
        <xs:element name="a1" minOccurs="0" type="xs:string" />
    </xs:sequence>
    <xs:attribute name="A" type="xs:string" />
</xs:complexType>
```

[**Notice**: the attribute declaration does not have a “use” attribute. Recall that an attribute declaration without a “use” attribute defaults to use="optional".]

Here is an element declared to be of that type:

```
<xs:element name="E2" type="C2" />
```

The content of element E2 can be any of these:

1. Pure text (xs:string)
2. Child element a1
3. Child element a1 plus text that surrounds a1
4. Pure text and attribute A
5. Child element a1 and attribute A
6. Child element a1, text that surrounds a1, and attribute A

Here are examples of the latter three.

4. E2 has pure text and attribute A:

```
<E2 A="foo">Hello World</E2>
```

5. E2 has a child a1 element and attribute A:

```
<E2 A="foo">
   <a1>hello world</a1>
</E2>
```

6. E2 has child element a1, text that surrounds a1, and attribute A:

```
<E2 A="foo">This is some text
   <a1>hello world</a1>
   that surrounds the element
</E2>
```

Restricting a complexType with mixed content and an attribute

Let C2 be the base type. It has three pieces of content that can be restricted away (i.e., removed):

(a) the optional child element, a1
(b) the text that surrounds the child element
(c) the optional attribute, A

Let's implement each of these restrictions.

(a) This complexType removes C2's markup:

```
<xs:complexType name="C2-sans-child-element">
    <xs:simpleContent>
        <xs:restriction base="C2">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:restriction>
    </xs:simpleContent>
</xs:complexType>
```

[Notice: the type does not declare the attribute. Nonetheless, it contains the attribute.]

[Notice: since its content is text, use simpleContent.]

Here I declare an element:

```
<xs:element name="E2-sans-child-element" type="C2-sans-child-element" />
```

Its only legal content is xs:string data and attribute A:

```
<E2-sans-child-element A="foo">Hello World</E2-sans-child-element>
```

[Important: the string data is unconstrained—it can be of any length and can contain any characters (Chinese, Arabic, Cyrillic, etc.)]

[Important: C2's markup can be restricted away only if the markup is optional. The child element a1 has minOccurs="0" and therefore can be removed.]

We can constrain the string data as shown here:

```
<xs:complexType name="C2-sans-child-element-constrained">
    <xs:simpleContent>
        <xs:restriction base="C2">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:maxLength value="10"/>
                    <xs:pattern value="[a-zA-Z0-9 ]*" />
                </xs:restriction>
            </xs:simpleType>
        </xs:restriction>
    </xs:simpleContent>
</xs:complexType>
```

The string data is now constrained to just 10 or less lower- and upper-case letters, digits and spaces.

(b) This complexType removes the text that can surround C2’s child element:

```
<xs:complexType name="C2-sans-text">
    <xs:complexContent>
        <xs:restriction base="C2">
            <xs:sequence>
                <xs:element name="a1" minOccurs="0" type="xs:string" />
            </xs:sequence>
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>
```

[Notice: it does not declare the attribute. Nonetheless, it has the attribute.]

[Notice: since its content is an element, use complexContent.]

Here I declare an element:

```
<xs:element name="E2-sans-text" type="C2-sans-text" />
```

Its only legal content is the child element a1 and the attribute A:

```
<E2-sans-text A="foo">
    <a1>hello world</a1>
</E2-sans-text>
```

(c) This complexType removes C2’s attribute:

```
<xs:complexType name="C2-sans-attribute" mixed="true">
    <xs:complexContent>
        <xs:restriction base="C2">
            <xs:sequence>
            <xs:element name="a1" minOccurs="0" type="xs:string" />
        </xs:sequence>
        <xs:attribute name="A" use="prohibited" type="xs:string" />
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
```

[Notice: here’s how to remove the attribute: use=“prohibited”.]

Here I declare an element:

```
<xs:element name="E2-sans-attribute" type="C2-sans-attribute" />
```

Its only legal content is mixed content:

```
  <E2-sans-attribute>This is some text
    <a1>hello world</a1>
    that surrounds the element
  </E2-sans-attribute>
```

Equivalent Types

The content of C2-sans-child-element is text plus attribute. So these two types are equivalent in terms of schema-valid data:

```
  <xs:complexType name="C2-sans-child-element">
    <xs:simpleContent>
      <xs:restriction base="C2">
        <xs:simpleType>
          <xs:restriction base="xs:string"/>
        </xs:simpleType>
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="C2-sans-child-element-equivalent">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="A" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
```

A bit different strategy is needed if the string data is constrained: the simpleType must be extracted, given a name, and made global. The equivalent of this:

```
  <xs:complexType name="C2-sans-child-element-constrained">
    <xs:simpleContent>
      <xs:restriction base="C2">
```

```
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:maxLength value="10"/>
                  <xs:pattern value="[a-zA-Z0-9 ]*" />
               </xs:restriction>
            </xs:simpleType>
         </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
```

is this pair:

```
   <xs:simpleType name="ABC">          <!-- Give the simpleType a unique name -->
      <xs:restriction base="xs:string">
         <xs:maxLength value="10"/>
         <xs:pattern value="[a-zA-Z0-9 ]*" />
      </xs:restriction>
   </xs:simpleType>

   <xs:complexType name="C2-sans-child-element-constrained-equivalent">
      <xs:simpleContent>
         <xs:extension base="ABC">
            <xs:attribute name="A" type="xs:string" />
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
```

[**Note**: XML Schema is not consistent with regard to constraining a simpleType in a complexType with simpleContent. A simpleType can occur within `complexType/simpleContent/restriction` but not within `complexType/simpleContent/extension`.]

The content of C2-sans-text is just the child element a1 plus the attribute. So these two types are equivalent in terms of schema-valid data:

```
   <xs:complexType name="C2-sans-text">
      <xs:complexContent>
         <xs:restriction base="C2">
            <xs:sequence>
               <xs:element name="a1" minOccurs="0" type="xs:string" />
            </xs:sequence>
         </xs:restriction>
      </xs:complexContent>
   </xs:complexType>

   <xs:complexType name="C2-sans-text-equivalent">
      <xs:sequence>
         <xs:element name="a1" minOccurs="0" type="xs:string" />
```

```
    </xs:sequence>
    <xs:attribute name="A" type="xs:string" />
</xs:complexType>
```

The content of C2-sans-attribute is mixed content. So these two types are equivalent in terms of schema-valid data:

```
<xs:complexType name="C2-sans-attribute" mixed="true">
    <xs:complexContent>
        <xs:restriction base="C2">
            <xs:sequence>
                <xs:element name="a1" minOccurs="0" type="xs:string" />
            </xs:sequence>
            <xs:attribute name="A" use="prohibited" type="xs:string" />
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>
```

```
<xs:complexType name="C2-sans-attribute-equivalent" mixed="true">
    <xs:sequence>
        <xs:element name="a1" minOccurs="0" type="xs:string" />
    </xs:sequence>
</xs:complexType>
```

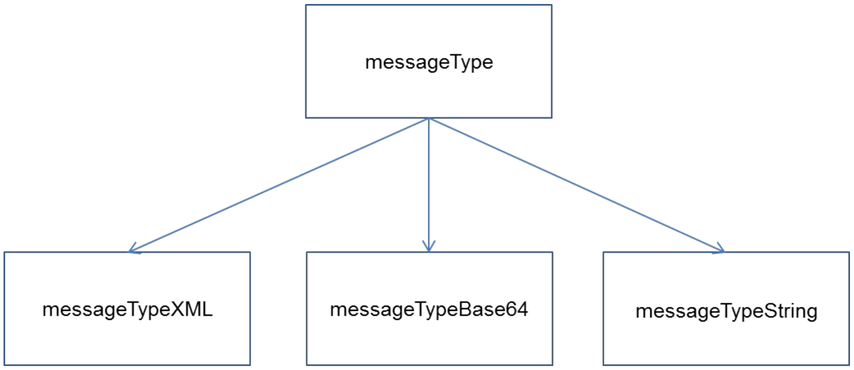
Real-world example of restricting a complexType with mixed content

This is a fabulous example from the XML Schema specification.

A complexType is created for representing various kinds of messages such as XML messages, base64 messages, and messages that just contain a string. A base type is created that is a complexType with mixed content and an attribute. Then several complexTypes are created which restrict the base type:

- A complexType for XML messages: restrict the base type by removing the ability to surround the child element.
- A complexType for base64 messages: restrict the base type by removing the child element and subtyping xs:string with xs:base64Binary.
- A complexType for string messages: restrict the base type by removing the child element.

Here is a graphic to illustrate the type hierarchy:


messageType
messageTypeXML
messageTypeBase64
messageTypeString

Here is the base type:

```
<xs:complexType name="messageType" mixed="true">
  <xs:sequence>
    <xs:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="kind">
    <xs:simpleType>
      <xs:union>
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="string"/>
            <xs:enumeration value="base64"/>
            <xs:enumeration value="binary"/>
            <xs:enumeration value="xml"/>
            <xs:enumeration value="XML"/>
          </xs:restriction>
        </xs:simpleType>
        <xs:simpleType>
          <xs:restriction base="xs:string"/>
        </xs:simpleType>
      </xs:union>
    </xs:simpleType>
  </xs:attribute>
  <xs:anyAttribute processContents="skip"/>
</xs:complexType>
```

Here is a message type specifically for XML messages:

```
<xs:complexType name="messageTypeXML">
  <xs:complexContent>
    <xs:restriction base="messageType">
      <xs:sequence>
        <xs:any processContents="strict"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

Here is a message type specifically for base64 messages:

```
<xs:complexType name="messageTypeBase64">
  <xs:simpleContent>
    <xs:restriction base="messageType">
      <xs:simpleType>
        <xs:restriction base="xs:base64Binary"/>
      </xs:simpleType>
```

```
        </xs:restriction>
    </xs:simpleContent>
</xs:complexType>
```

Here is a message type specifically for string messages:

```
<xs:complexType name="messageTypeString">
    <xs:simpleContent>
        <xs:restriction base="messageType">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:restriction>
    </xs:simpleContent>
</xs:complexType>
```

My XML Schema for complexTypes shown at the top

Below is my XML Schema for the complexTypes shown at the beginning of this document

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <!-- **************************** -->
    <!--     Mixed content          -->
    <!-- **************************** -->

    <xs:complexType name="C1" mixed="true">
        <xs:sequence>
            <xs:element name="a1" minOccurs="0" type="xs:string" />
        </xs:sequence>
    </xs:complexType>

    <!-- Remove C1's child element a1 to leave just the text -->

    <xs:complexType name="C1-sans-child-element">
        <xs:simpleContent>
            <xs:restriction base="C1">
                <xs:simpleType>
                    <xs:restriction base="xs:string"/>
                </xs:simpleType>
            </xs:restriction>
        </xs:simpleContent>
    </xs:complexType>

    <!-- Remove C1's text to leave just the child element a1 -->

    <xs:complexType name="C1-sans-text">
```

```
   <xs:complexContent>
      <xs:restriction base="C1">
         <xs:sequence>
            <xs:element name="a1" minOccurs="0" type="xs:string" />
         </xs:sequence>
      </xs:restriction>
   </xs:complexContent>
</xs:complexType>

<!-- This is equivalent to C1-sans-child-element: it has just text -->

<xs:simpleType name="C1-sans-child-element-equivalent">
   <xs:restriction base="xs:string" />
</xs:simpleType>

<!-- This is equivalent to C1-sans-text: it just has child element a1 -->

<xs:complexType name="C1-sans-text-equivalent">
   <xs:sequence>
      <xs:element name="a1" minOccurs="0" type="xs:string" />
   </xs:sequence>
</xs:complexType>

<!-- **************************** -->
<!-- Mixed content plus attribute -->
<!-- **************************** -->

<xs:complexType name="C2" mixed="true">
   <xs:sequence>
      <xs:element name="a1" minOccurs="0" type="xs:string" />
   </xs:sequence>
   <xs:attribute name="A" type="xs:string" />
</xs:complexType>

<!-- Remove C2's child element a1 to leave text plus attribute -->

<xs:complexType name="C2-sans-child-element">
   <xs:simpleContent>
      <xs:restriction base="C2">
         <xs:simpleType>
            <xs:restriction base="xs:string"/>
         </xs:simpleType>
      </xs:restriction>
   </xs:simpleContent>
</xs:complexType>
```

```
<!-- Remove the text that surrounds C2's child element to leave element plus attribute -->

<xs:complexType name="C2-sans-text">
   <xs:complexContent>
      <xs:restriction base="C2">
         <xs:sequence>
            <xs:element name="a1" minOccurs="0" type="xs:string" />
         </xs:sequence>
      </xs:restriction>
   </xs:complexContent>
</xs:complexType>

<!-- Remove C2's attribute to leave mixed content -->

<xs:complexType name="C2-sans-attribute" mixed="true">
   <xs:complexContent>
      <xs:restriction base="C2">
         <xs:sequence>
            <xs:element name="a1" minOccurs="0" type="xs:string" />
         </xs:sequence>
         <xs:attribute name="A" use="prohibited" type="xs:string" />
      </xs:restriction>
   </xs:complexContent>
</xs:complexType>

<!-- This is equivalent to C2-sans-child-element: it has text plus attribute -->

<xs:complexType name="C2-sans-child-element-equivalent">
   <xs:simpleContent>
      <xs:extension base="xs:string">
         <xs:attribute name="A" type="xs:string" />
      </xs:extension>
   </xs:simpleContent>
</xs:complexType>

<!-- This is equivalent to C2-sans-text: it has element plus attribute -->

<xs:complexType name="C2-sans-text-equivalent">
   <xs:sequence>
      <xs:element name="a1" minOccurs="0" type="xs:string" />
   </xs:sequence>
   <xs:attribute name="A" type="xs:string" />
</xs:complexType>

<!-- This is equivalent to C2-sans-attribute: it has mixed content (no attribute) -->
```

```
   <xs:complexType name="C2-sans-attribute-equivalent" mixed="true">
      <xs:sequence>
         <xs:element name="a1" minOccurs="0" type="xs:string" />
      </xs:sequence>
   </xs:complexType>

   <xs:element name="Test">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="E1" type="C1" maxOccurs="unbounded" />
            <xs:element name="E1-sans-child-element" type="C1-sans-child-element" />
            <xs:element name="E1-sans-text" type="C1-sans-text" />
            <xs:element name="E1-sans-child-element-equivalent" type="C1-sans-child-element-equivalent" />
            <xs:element name="E1-sans-text-equivalent" type="C1-sans-text-equivalent" />

            <xs:element name="E2" type="C2" maxOccurs="unbounded" />
            <xs:element name="E2-sans-child-element" type="C2-sans-child-element" />
            <xs:element name="E2-sans-text" type="C2-sans-text" />
            <xs:element name="E2-sans-attribute" type="C2-sans-attribute" />
            <xs:element name="E2-sans-child-element-equivalent" type="C2-sans-child-element-equivalent" />
            <xs:element name="E2-sans-text-equivalent" type="C2-sans-text-equivalent" />
            <xs:element name="E2-sans-attribute-equivalent" type="C2-sans-attribute-equivalent" />
         </xs:sequence>
      </xs:complexType>
   </xs:element>

</xs:schema>
```

Here is a sample instance document:

```
<?xml version="1.0" encoding="UTF-8"?>
<Test xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="A.xsd">

   <E1>This is some text
      <a1>hello world</a1>
      that surrounds the element
   </E1>

   <E1-sans-child-element>Hello World</E1-sans-child-element>

   <E1-sans-text>
      <a1>hello world</a1>
   </E1-sans-text>

   <E1-sans-child-element-equivalent>Hello World</E1-sans-child-element-equivalent>
```

```xml
  <E1-sans-text-equivalent>
    <a1>hello world</a1>
  </E1-sans-text-equivalent>

  <E2 A="foo">This is some text
    <a1>hello world</a1>
    that surrounds the element
  </E2>

  <E2-sans-child-element A="foo">Hello World</E2-sans-child-element>

  <E2-sans-text A="foo">
    <a1>hello world</a1>
  </E2-sans-text>

  <E2-sans-attribute>This is some text
    <a1>hello world</a1>
    that surrounds the element
  </E2-sans-attribute>

  <E2-sans-child-element-equivalent A="foo">Hello World</E2-sans-child-element-equivalent>

  <E2-sans-text-equivalent A="foo">
    <a1>hello world</a1>
  </E2-sans-text-equivalent>

  <E2-sans-attribute-equivalent>This is some text
    <a1>hello world</a1>
    that surrounds the element
  </E2-sans-attribute-equivalent>
</Test>
```