http://www.sqlservercentral.com/articles/XML/61261/
Printed 2007/10/17 04:57PM

XML Workshop VIII - Custom Types and Inheritance

By Jacob Sebastian, 2007/10/08

Introduction

This is the 8th installment of my *XML Workshop* which aims at explaining the details of working with *XML* in *TSQL*. You can find the the other articles in this series here. In the previous articles, we have seen how to read values from *XML* variables and columns, generate *XML* structures using *FOR XML* with *AUTO, RAW, PATH* and *EXPLICIT*.

The last two articles were focusing on *TYPED XML* and had been explaining how to generate an *XML* schema that can be used to validate the *XML* data. We have seen how to validate the existence of *elements* and *attributes*, how to validate *data types* and how to apply *restrictions* on the actual data.

XML Schemas usually are very complex and confusing. The more validation and rules you apply, the more complex the schema will become. This article will present a couple of approaches that will help you to make your schemas less complex and easier to read and understand.

Creating Custom Types

One of the ways to simplify your schema is by using *Custom Types*. If an *element* is having very complex validation rules, it will make more sense to move it to a *Custom Type* and then add an element of the custom type in the *schema* definition. *Custom types* are more useful when you need to reuse the validations on different elements. For example, the validation for *Phone number* can be used for other elements like *Phone-Home, Phone-Office, Fax* etc.

Let us look at an example which uses a *custom type*. I am updating the *XML* schema that we used in the previous example. The new version of the XML schema has a *Custom Type*: *AgeType*. Note that the validation rules do not change. The *XML* structure expected by the schema is the same. What we changed is only the way we defined it. [See sql1.sql and schema1.xml in the code download]

```
DROP XML SCHEMA COLLECTION EmployeeSchema
GO
/*
    A schema may look very complex if there are lots of elements,
    validations and restrictions. Many of the schemas that you might
    use in a production environment would be very long and complex.

    To simplify the schema development and to make it easier to understand
    and manage, you can define your own custom types. These types can be used
    as children of a "complexType".

    The following example is using the Schema that we defined in the previous
    session. I have modified the previous schema, to demonstrate the usage of
    a custom type. In this version, I have defined a type named "AgeType".
    You can find the definition of the type at the bottom of the schema
definition.
    Note that I have defined an element which is of type "AgeType" in the
schema
    definition.

    Use of custom "types" will make the schema development simpler and easier.
    It helps you to re-use the validation rules defined, in multiple instances
    of the type.
```

```
*/
CREATE XML SCHEMA COLLECTION EmployeeSchema AS '
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Employee">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="FullName" type="xs:string" />
        <xs:element name="Age" type="AgeType" />
        <xs:element name="Nationality">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="Indian" />
              <xs:enumeration value="British" />
              <xs:enumeration value="American" />
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="Gender">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:pattern value="male|female" />
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="Salary">
          <xs:simpleType>
            <xs:restriction base="xs:decimal">
              <xs:fractionDigits value="2" />
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="EmployeeNumber" use="required" >
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            <xs:totalDigits value="6" />
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="LoginName" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:minLength value="6" />
            <xs:maxLength value="8" />
            <xs:pattern value="([a-z])*" />
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="AgeType">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="18" />
      <xs:maxInclusive value="65" />
    </xs:restriction>
  </xs:simpleType>
  </xs:schema>
'
GO

/*
The following is a correct XML value that validates with the above SCHEMA. The
XML value is the same that we used in the previous session. The only change
that we did in this version is a restructuring of the schema, which does not
affect the validation rules.
*/
DECLARE @emp AS XML(EmployeeSchema)
SET @emp = '
<Employee EmployeeNumber="1001" LoginName="jacobs" >
    <FullName>Jacob</FullName>
    <Age>30</Age>
    <Nationality>Indian</Nationality>
```

```
    <Gender>male</Gender>
    <Salary>10000.00</Salary>
</Employee>
'
```

Probably it might be a good idea to move all the elements that we have to *Custom Types*. Let us do it. [see sql2.sql and schema2.xml in the code download]

```
DROP XML SCHEMA COLLECTION EmployeeSchema
GO
/*
Here is the final version of the schema. I have broken all the elements into
various
custom types.
*/
CREATE XML SCHEMA COLLECTION EmployeeSchema AS '
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Employee">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="FullName" type="xs:string" />
        <xs:element name="Age" type="AgeType" />
        <xs:element name="Nationality" type="NationalityType" />
        <xs:element name="Gender" type="GenderType" />
        <xs:element name="Salary" type="SalaryType" />
      </xs:sequence>
      <xs:attribute name="EmployeeNumber" use="required"
type="EmployeeNumberType" />
      <xs:attribute name="LoginName" use="required" type="LoginNameType" />
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="AgeType">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="18" />
      <xs:maxInclusive value="65" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="NationalityType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Indian" />
      <xs:enumeration value="British" />
      <xs:enumeration value="American" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="GenderType">
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="SalaryType">
    <xs:restriction base="xs:decimal">
      <xs:fractionDigits value="2" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="EmployeeNumberType" >
    <xs:restriction base="xs:integer">
      <xs:totalDigits value="6" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="LoginNameType">
    <xs:restriction base="xs:string">
      <xs:minLength value="6" />
      <xs:maxLength value="8" />
      <xs:pattern value="([a-z])*" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
'
GO

/*
The following is a correct XML value that validates with the above SCHEMA. Note
```

```
that the changes in the schema structure did not affect the validation rules.
*/
DECLARE @emp AS XML(EmployeeSchema)
SET @emp = '
<Employee EmployeeNumber="1001" LoginName="jacobs" >
    <FullName>Jacob</FullName>
    <Age>30</Age>
    <Nationality>Indian</Nationality>
    <Gender>male</Gender>
    <Salary>10000.00</Salary>
</Employee>
'
```

Inheriting from a Custom Type

Anyone who understands *OOPS* would also understand the power and freedom that inheritance can give to a class. It is the same with *Custom Types* in a *SCHEMA* definition. It supports inheritance. You can derive other custom types from a parent type. Use of inheritance can make your *SCHEMA* definition even simpler and easier to understand. I have put up a quick example that shows how to inherit from a *Custom Type* within a Schema definition.[See sql3.sql and schema3.xml in the code download]

```
DROP XML SCHEMA COLLECTION EmployeeSchema
GO
/*
We have just seen how to define custom types. This example explains
how to create a derived type which inherits the rules defined in a parent
type.

The following SCHEMA defines a type "EmployeeType". "ProjectManagerType"
and "SupportEngineerType" inherit from "EmployeeType". Each of the
derived types has their own elements and can have own validation rules.
All the elements, properties and validation rules defined in the parent
type will be available to the derived type too.
*/
CREATE XML SCHEMA COLLECTION EmployeeSchema AS '
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Team">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ProjectManager" type="ProjectManagerType" />
        <xs:element name="SupportEngineer" type="SupportEngineerType" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="EmployeeType"> <!-- This is our base class -->
    <xs:sequence>
      <xs:element name="FullName" type="xs:string"/>
      <xs:element name="Department" type="xs:string"/>
      <xs:element name="ReportsTo" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ProjectManagerType"> <!-- This is a derived class -->
    <xs:complexContent>
      <xs:extension base="EmployeeType"> <!-- Note the usage of "base" -->
        <xs:sequence>
          <xs:element name="ProjectName" type="xs:string"/>
          <xs:element name="Technology" type="xs:string"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="SupportEngineerType"><!-- This is a derived class -->
    <xs:complexContent>
      <xs:extension base="EmployeeType"><!-- Note the usage of "base" -->
        <xs:sequence>
          <xs:element name="Phone" type="xs:string"/>
          <xs:element name="Skype" type="xs:string"/>
          <xs:element name="IM" type="xs:string"/>
        </xs:sequence>
```

```
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
'
GO

/*
Here is the XML that validates with the above schema. Note that we have
two elements: ProjectManager and SupportEngineer. Both of them have all
the elements defined in the EmployeeType: FullName, Department and
ReportsTo.
*/

DECLARE @emp AS XML(EmployeeSchema)
SET @emp = '
<Team>
    <ProjectManager>
        <FullName>Robert V</FullName>
        <Department>Software</Department>
        <ReportsTo>CEO</ReportsTo>
        <ProjectName>Virtual Earth</ProjectName>
        <Technology>ASP.NET</Technology>
    </ProjectManager>
    <SupportEngineer>
        <FullName>Michael M</FullName>
        <Department>Tech Support</Department>
        <ReportsTo>Robert V</ReportsTo>
        <Phone>999-999-9999</Phone>
        <Skype>whoami</Skype>
        <IM>whoami@whereami.com</IM>
    </SupportEngineer>
</Team>
'
```

Conclusions

This article explains how to create *custom types* and create types that *inherit* from other types. Usage of *custom types* can make the *XML* schema less complex and easier to understand and manager.
