

Nach Schema F

Zur Beschreibung der Struktur von XML-Dokumenten existieren mehrere Sprachen. dotnetpro stellt in dieser Serie die drei wichtigsten vor. Der W3C-Standard XML Schema macht den Anfang.

Auf einen Blick



Jürgen Bayer arbeitet seit mehr als 15 Jahren als freiberuflicher .NET-Consultant, Softwarearchitekt, IT-Trainer und Autor von Programmierbüchern wie dem *C# 2008 Codebook*. Sie erreichen ihn unter jb@juergen-bayer.net.

Inhalt

- ➔ Der Nutzen von XML-Schemas.
- ➔ Ein fundierter Überblick zu XML Schema 1.0.
- ➔ Vor- und Nachteile von XML Schema 1.0.

Serie

1. Grundlagen der XML-Validierung mit XML Schema 1.0.
2. Geschäftsregeln und bedingter Inhalt in XML-Schemas.
3. XML-Validierung mit Relax NG.



dnpCode

A1004Schematron

Bei der Verarbeitung von XML-Dokumenten gibt es zwei Wege: Simple Lösungen lesen ein Dokument ein und reagieren auf Ausnahmen für den Fall, dass erwartete Elemente oder Attribute nicht vorhanden sind oder einen falschen Typ aufweisen. Bessere Lösungen setzen ein Schema ein, um XML-Dokumente vor der Verarbeitung zu prüfen.

Die Prüfung auf Einhaltung eines Schemas vereinfacht die Programmierung. Entspricht ein XML-Dokument dem Schema, kann ein Programm sich darauf verlassen, dass das Dokument problemlos verarbeitet werden kann. Ein Schema stellt idealerweise zumindest sicher, dass alle erwarteten Elemente und Attribute enthalten sind, dass diese einen erwarteten Typ besitzen und lediglich erlaubte Werte speichern.

XML-Schemas dienen aber auch als Vertrag zwischen Kommunikationspartnern. Partner ermitteln über das Schema die Struktur und den erwarteten Inhalt der zu erzeugenden Dokumente. Außerdem können sie Dokumente vor der Weitergabe prüfen. Schemas vereinfachen auf diese Weise die Kommunikation über XML.

Gute Gründe für den Einsatz von XML-Schemas gibt es also genug. Um die Frage zu klären, welche der bekannten Schemasprachen die geeignete ist, stellt dotnetpro in dieser Serie die drei wichtigsten Schemasprachen vor. Den Anfang macht die vom W3C standardisierte Sprache XML Schema Definition Language (XSD), auch kurz als XML Schema bezeichnet. Da .NET 3.5 und .NET 4.0 die neueste Version 1.1 [1] [2] noch nicht unterstützen, konzentriert sich der erste Teil der Serie auf die Version 1.0 [3] [4].

Das Beispiel

Die grundlegenden Elemente von XML Schema werden an einem XML-Dokument erläutert, das Projektdaten verwaltet (Listing 1). Listing 2 zeigt ein einfaches Schema für ein solches Dokument.

Dieses Schema kann recht einfach von oben nach unten gelesen werden: Ein dem Schema entsprechendes XML-Dokument muss ein Wurzelement *projects* enthalten. *projects* ist ein komplexer Typ, der eine Sequenz von *project*-Elementen enthält. Die maximale Anzahl der enthaltenen *project*-Elemente ist unbegrenzt (*unbounded*). Ein *project*-Element ist ebenfalls ein komplexer Typ. Dieser muss die Elemente *name*

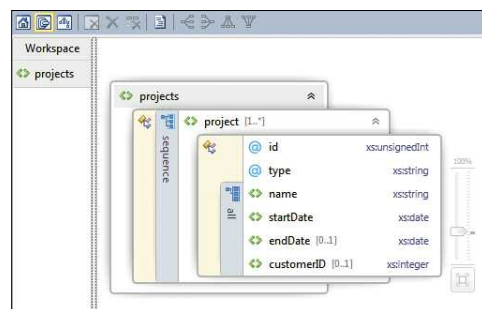
Listing 1

Das XML-Beispiel für die grundlegenden Schemaelemente.

```
<?xml version="1.0" ?>
<projects>
  <project id="1000" type="business">
    <name>XML-Schema-Artikel</name>
    <startDate>2009-12-01</startDate>
    <endDate>2009-12-08</endDate>
    <customerID>1001</customerID>
  </project>
  <project id="1001" type="private">
    <name>Website</name>
    <startDate>2010-01-01</startDate>
  </project>
</projects>
```

und *startDate* enthalten. Die Elemente *endDate* und *customerID* können, müssen aber nicht vorhanden sein, da bei beiden *minOccurs="0"* gesetzt ist. *name* muss einen String verwalten, *startDate* und *endDate* ein Datum und *customerID* einen Integerwert ohne Vorzeichen. Ein *project*-Element muss außerdem die Attribute *id* und *type* besitzen, wobei der Wert von *id* ein Integerwert ohne Vorzeichen sein muss.

Ein solches Schema können Sie recht einfach über Visual Studio erzeugen. Öffnen Sie dazu eine XML-Datei, und wählen Sie im XML-Menü den Befehl *Create Schema* beziehungsweise *Schema erstellen*. Das erzeugte Schema muss aber in den meisten Fällen nachbearbeitet werden. Beim Bearbeiten ist der Schemaeditor von Visual Studio



[Abb. 1] Die Modellsicht des Beispielschemas im Visual-Studio-2010-Schemaeditor.

sehr hilfreich, da IntelliSense jeweils nur die syntaktisch möglichen Elemente anbietet.

Ab VS 2010 ist auch ein visueller Schemaeditor integriert. Abbildung 1 zeigt das Schema aus Listing 2 in diesem Editor.

Beachten Sie bitte, dass dieses Schema das *id*-Attribut nicht auf Eindeutigkeit einschränkt, für *name* keinen Wert erzwingt und den Inhalt des *type*-Attributs nicht einschränkt. Einschränkungen dieser Art werden noch erläutert. Einige Einschränkungen können allerdings mit XML Schema 1.0 nicht abgebildet werden:

- Kookkurrenz-Einschränkungen, bei denen die Werte in Elementen oder Attributen sich auf andere Elemente/Attribute oder auf Daten in separaten Dokumenten beziehen. Ein Beispiel ist, dass *endDate* größer/gleich *startDate* sein muss.
- Bedingter Inhalt, bei dem der Inhalt von Elementen von Werten in anderen Elementen oder Attributen abhängig ist. Ein Beispiel ist, dass das *customerID*-Element für *project*-Elemente mit *type="business"* vorhanden sein muss.

Einschränkungen dieser Art werden im nächsten Teil dieser Serie behandelt.

XML-Schemas in .NET

.NET bietet Unterstützung für XML-Schemas über die *XmlValidatingReader*-Klasse und die Erweiterungsmethode *Validate*, die für einige Klassen des XDOM wie *XElement* gilt. Da ein *XmlValidatingReader* auch zum Einlesen eines XML-Dokuments in ein *XmlDocument*, *XDocument* oder *XElement* eingesetzt werden kann, können Sie mit allen XML-Technologien von .NET validierend lesen. Einen *XmlValidatingReader* erzeugen Sie über die Überladung der *Create*-Methode der *XmlReader*-Klasse, der am Argument *settings* ein *XmlReaderSettings*-Objekt übergeben wird. Über dieses Objekt definieren Sie, dass ein Schema berücksichtigt werden soll, und geben dieses an, wenn es nicht in das XML-Dokument integriert ist. Um zu verhindern, dass das Lesen bei der ersten aufgefundenen Verletzung des Schemas mit einer Ausnahme abgebrochen wird, können Sie das *ValidationEventHandler*-Ereignis zuweisen, siehe dazu Listing 3.

Einfache und komplexe Typen

Das Beispiel in Listing 2 enthält die meisten grundlegenden XML-Schemaelemente. Das Wurzelement ist *xs:schema*. Die Schemaelemente sind dem Namensraum <http://www.w3.org/2001/XMLSchema> zuge-

Listing 2

Schema für das einfache XML-Beispiel.

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="projects">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="project" maxOccurs="unbounded">
          <xs:complexType>
            <xs:all>
              <xs:element name="name" type="xs:string" />
              <xs:element name="startDate" type="xs:date" />
              <xs:element name="endDate" type="xs:date" minOccurs="0" />
              <xs:element name="customerID" type="xs:integer" minOccurs="0" />
            </xs:all>
            <xs:attribute name="id" type="xs:unsignedInt" use="required" />
            <xs:attribute name="type" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Listing 3

Einlesen eines XML-Dokuments mit Schemaprüfung.

```
private List<string> validationErrors = new List<string>();
private List<string> validationWarnings = new List<string>();

...

var settings = new XmlReaderSettings();
settings.ValidationType = ValidationType.Schema;
settings.ValidationEventHandler += ValidationEventHandler;
string schemaUri = "...";
settings.Schemas.Add(null, schemaUri);

string xmlUri = "...";
XmlReader xmlReader = XmlReader.Create(xmlUri, settings);

XDocument doc = XDocument.Load(xmlReader);

// Auswerten der Fehler in validationErrors und der Warnungen in validationWarnings
if (validationErrors.Count > 0)
{
  ...
}

...

private void ValidationEventHandler(object sender, ValidationEventArgs e)
{
  if (e.Severity == XmlSeverityType.Error)
  {
    this.validationErrors.Add(e.Message);
  }
  else
  {
    this.validationWarnings.Add(e.Message);
  }
}
```

Listing 4

Deklaration eines komplexen Typs mit einfachem Inhalt.

```
<xs:element name="name">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="language" type="xs:language" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Tabelle 1

Die wichtigsten vordefinierten einfachen XML-Schematypen.

Typ	Beschreibung
xs:string, xs:normalizedString, xs:token	String mit allen für XML 1.0 gültigen Zeichen. Bei <i>xs:normalizedString</i> werden bei der Verarbeitung Whitespace-Zeichen durch Leerzeichen ersetzt. Bei <i>xs:token</i> werden zusätzlich Whitespace-Zeichen am Anfang und Ende des Strings entfernt.
xs:long, xs:int, xs:short, xs:byte, xs:unsignedLong, xs:unsignedInt, xs:unsignedShort, xs:unsignedByte	Ganzzahlen mit den von .NET bekannten Wertebereichen.
xs:float, xs:double	Fließkommawerte nach IEEE. <i>xs:float</i> ist wie gewohnt 32 Bit, <i>xs:double</i> 64 Bit groß. Zulässige Literale sind normale Zahlenwerte mit Punkt als Dezimaltrennzeichen, die wissenschaftliche Schreibweise und die Literale <i>+0</i> , <i>-0</i> , <i>INF</i> , <i>-INF</i> und <i>NaN</i> für die speziellen Fließkommawerte.
xs:boolean	Boole'sche Werte in Form der Literale <i>true</i> , <i>false</i> , <i>1</i> oder <i>0</i> .
xs:dateTime, xs:date, xs:time, xs:gMonthDay, xs:gYearMonth	Datums- oder Zeitwerte beziehungsweise Teile davon. Das Literal im XML-Element folgt dem Muster <i>CCYY-MM-DDThh:mm:ss</i> . Das <i>T</i> ist ein fester Trenner zwischen Datums- und Zeitteil. Bei <i>xs:dateTime</i> müssen alle Teile angegeben werden, bei den anderen Typen nur die entsprechenden Teile. Die Zeitzone kann als UTC in Form des Buchstabens <i>Z</i> oder als Offset zu UTC in der Form <i>+hh:mm</i> oder <i>-hh:mm</i> angehängt werden.

ordnet. Dieser Namensraum wird üblicherweise dem Präfix *xs* zugeordnet.

Das Element *xs:element* beschreibt ein XML-Element. Das erforderliche Attribut *name* gibt den Namen an. Für Elemente, deren Inhalt lediglich ein Textknoten ist, gibt das Attribut *type* den erwarteten Typ an. Der einfache Typ eines solchen Elements kann einer der in XML Schema vordefinierten oder ein davon abgeleiteter sein.

Elemente, die Kindelemente und/oder Attribute enthalten, sind komplexe Typen. Deren Inhalt wird über *xs:complexType* definiert. Die Elemente *projects* und *project* in Listing 2 sind komplexe Typen mit Kindelementen. Für die Kindelemente gibt das optionale Attribut *minOccurs* an, wie oft das XML-Element mindestens enthalten sein muss. Der Defaultwert ist 1. *maxOccurs* gibt die maximale Anzahl des XML-Elements an. Der Defaultwert ist ebenfalls 1. Die Definition des *endDate*-Elements setzt *minOccurs* auf 0, um zu erreichen, dass dieses Element auch weggelassen werden kann. Der besondere Wert *unbounded*, der mit Einschränkungen in XML Schema 1.0 für *max-*

Occurs gültig und beim *project*-Element angegeben ist, bedeutet, dass die maximale Anzahl unbegrenzt ist. Die Kindelemente eines komplexen Typs werden den folgenden *Kompositoren* untergeordnet:

- *xs:sequence*: Legt fest, dass alle untergeordnet angegebenen XML-Elemente in der im *xs:element*-Element über *minOccurs* und *maxOccurs* angegebenen Anzahl enthalten sein müssen. Die Reihenfolge der Elemente ist dabei festgelegt.
- *xs:all*: Dieses Element nimmt, ähnlich wie *xs:sequence*, ebenfalls Kindelemente auf. Es unterscheidet sich von *xs:sequence* darin, dass die Reihenfolge der Elemente nicht festgelegt ist und *minOccurs* und *maxOccurs* in XML Schema 1.0 eingeschränkt sind.
- *xs:choice*: Legt fest, dass genau eines der untergeordnet angegebenen XML-Elemente enthalten sein muss.

xs:sequence und *xs:choice* erlauben für die enthaltenen Elemente in den Attributen *minOccurs* und *maxOccurs* einen Integerwert größer/gleich 0 und in *maxOccurs*

zusätzlich *unbounded*. *minOccurs* ist für *xs:all* auf 0 oder 1 eingeschränkt. XML Schema 1.0 erlaubt für *xs:all* in *max-Occurs* lediglich den Wert 1. Dies macht in XML Schema 1.0 die Definition solcher Schemas sehr schwierig, die Kindelemente in beliebiger Reihenfolge, jedoch in einer Anzahl größer als 1 zulassen. XML Schema 1.1 lässt aber für *xs:all* in *maxOccurs* Werte größer als 1 und *unbounded* zu.

Attribute werden immer am Ende eines komplexen Typs über *xs:Attribute* deklariert. Wie bei einfachen Elementen gibt *type* den Typ des Attributs an. Das optionale Attribut *use* bestimmt mit *required*, dass das Attribut erforderlich ist. Der Wert *optional* macht das Attribut optional. Sie sollten beachten, dass dies die Voreinstellung ist. Der dritte Wert *prohibited* ist für abgeleitete Typen vorgesehen und verbietet ein geerbtes Attribut. Komplexe Typen mit einfachem Inhalt und Attributen werden etwas anders deklariert als komplexe Typen mit Kindelementen. Der einfache Inhalt wird über *xs:simpleContent* beschrieben. Der Inhalt dieses Elements ist eine Ableitung eines einfachen Typs, der um Attribute erweitert wird. Listing 4 zeigt, wie das *name*-Element erweitert werden kann.

Einfache Typen und Ableitungen

XML Schema stellt eine Vielzahl einfacher Typen zum direkten Einsatz in einfachen Elementen oder Attributen oder als Basis für eigene Typen zur Verfügung. Der Inhalt von Elementen oder Attributen, die über das *type*-Attribut mit einem einfachen Typ versehen sind, muss diesem entsprechen, um gültig zu sein. Tabelle 1 zeigt eine Übersicht über die wichtigsten Typen. Eine vollständige Liste der XML-Schema-1.0-Typen erhalten Sie bei [5].

Für den Fall, dass die durch die vordefinierten Typen gegebene Werteschränkung nicht ausreicht, sieht XML Schema das Ableiten von einfachen Typen vor. Beim Ableiten gibt es drei Möglichkeiten:

- Die Ableitung über eine Einschränkung schränkt die möglichen Literale oder die dem Basistyp entsprechenden Werte ein.
- Bei der Ableitung über eine Liste wird der Basistyp erweitert, um eine Whitespace-begrenzte Liste der Literale anzugeben, die der Basistyp zulässt.
- Die Ableitung über eine Vereinigung erzeugt einen Typ, der den lexikalischen Raum mehrerer Basistypen zulässt.

Dieser Artikel behandelt nur die sehr häufig eingesetzte Ableitung über eine Ein-

schränkung. Eine solche wird für einen einfachen Typ über das *xs:simpleType*-Element definiert, der ein *xs:restriction*-Element enthält. Dessen *base*-Attribut gibt den Basistyp an. Im *xs:restriction*-Element bringen Sie sogenannte Facetten unter. Eine Facette beschreibt die eigentliche Einschränkung über ihr *value*-Attribut. Welche Facetten verwendet werden können, hängt vom Basistyp ab. Tabelle 2 beschreibt die wichtigsten.

Die meisten Facetten erklären sich von selbst. *xs:enumeration* und *xs:minInclusive* werden in Listing 5 angewandt. In diesem Artikel bleibt für *xs:pattern* kein Platz, mehr dazu unter [6].

Für das *projects*-Dokument wäre es angebracht, den Wert des *id*-Attributs auf Zahlen größer als 0 einzuschränken. Das *type*-Attribut sollte nur die Werte *business* und *private* zulassen. Über eine Ableitung mit Einschränkung ist eine solche Aufgabe schnell gelöst, wie Listing 5 zeigt.

Beim Einsatz von Facetten sollten Sie beachten, dass mehrere Facetten gleichen Typs über eine *Oder*-Verknüpfung verbunden werden. In Listing 5 bedeuten die Facetten in der Restriktion des *type*-Attributs, dass entweder *private* oder *business* angegeben werden kann. Facetten unterschiedlichen Typs werden hingegen über ein logisches *Und* verknüpft. Ein Anwendungsbeispiel dafür finden Sie im Beispielprojekt.

Matroschka- und flache Schemen

Ein Schema wie in Listing 2 wird auch als Matroschka-Schema (Russian-Doll-Schema) bezeichnet. Grund dafür ist, dass die Typen der einzelnen Elemente lokal unterhalb ihrer Elternelemente deklariert sind.

Es gibt jedoch auch die Möglichkeit, Elementtypen global zu deklarieren und diese in Elementen oder Attributen als Typ zu referenzieren. Globale Elementtypen liegen direkt unter *xs:schema*. Handelt es sich um einen komplexen Typ, wird dieser in entsprechenden Elementen über das *ref*-Attribut referenziert. Ein einfacher Typ wird wie gewohnt über das *type*-Attribut angegeben.

Das bisherige Schema könnte zum Beispiel auch so aufgebaut sein wie in Listing 6.

Der Vorteil eines flachen Schemas ist, dass globale Typen an beliebigen Stellen im eigentlichen Schema eingesetzt werden können. Das hat stets dann Sinn, wenn ein benutzerdefinierter Typ für unterschiedliche Elemente verwendet werden soll. In einigen Fällen fördern globale Typen auch die Lesbarkeit oder sind implementierungsbedingt notwendig, wie zum Beispiel bei rekursiven Schemata.

Tabelle 2

Die wichtigsten Facetten für Ableitungen über eine Einschränkung.

Facette	Beschreibung
<i>xs:enumeration</i>	Definiert einen oder mehrere mögliche Werte. Diese Facette kann auf String-, Zahl- und Datumstypen angewendet werden.
<i>xs:length</i> , <i>xs:minLength</i> , <i>xs:maxLength</i>	Bestimmen die Anzahl der Zeichen in einem Stringtyp.
<i>xs:pattern</i>	Erlaubt es mit regulären Ausdrücken zu arbeiten. Die Syntax ist prinzipiell dieselbe wie die von Perl, auf welcher auch die Syntax der regulären Ausdrücke in .NET basiert. Sie können <i>xs:pattern</i> auf String-, Zahl- und Datumstypen anwenden.
<i>xs:minExclusive</i> , <i>xs:minInclusive</i> , <i>xs:maxExclusive</i> , <i>xs:maxInclusive</i>	Definieren Minimal- und Maximalwerte für Zahl- und Datumstypen.

Listing 5

Einschränkungen der Attribute des project-Elements.

```
<xs:element name="project" maxOccurs="unbounded">
  <xs:complexType>
    <xs:all>
      <xs:element name="name" type="xs:string" />
      <xs:element name="startDate" type="xs:date" />
      <xs:element name="endDate" type="xs:date" minOccurs="0" />
      <xs:element name="customerID" type="xs:unsignedInt" minOccurs="0" />
    </xs:all>
    <xs:attribute name="id" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:unsignedInt">
          <xs:minInclusive value="1"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="type" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="private"/>
          <xs:enumeration value="business"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
```

Nullwerte

Die Typen *xs:string*, *xs:normalizedString*, *xs:token*, *xs:anyURI*, *xs:hexBinary*, *xs:base64Binary* und *xs:anySimpleType* lassen Leerwerte zu. Alle anderen Typen erlauben zunächst keine Leer- und auch keine Nullwerte. XML Schema unterscheidet zwischen Elementen, die einen leeren Inhalt besitzen, und Elementen, die Nullwerte speichern. Letztere müssen mit dem Attribut *xsi:nil="true"* gekennzeichnet sein. *xsi* ist dabei das übliche Präfix des XML-Schema-Instanz-Namensraums <http://www.w3.org/2001/XMLSchema-instance>. Dieser enthält Elemente wie *xsi:nil*, die in Schemata vorausgesetzt und in XML-Dokumenten eingesetzt werden.

Um im Schema Nullwerte zuzulassen, setzen Sie das *nillable*-Attribut eines *xs:elements*-Elements auf *true*. Attribute erlauben keine Nullwerte. Im Beispielprojekt hat das Zulassen von Nullwerten für das *startDate*-Element Sinn. Wenn auch *minOccurs* weggelassen oder auf 1 gesetzt wird, wird erreicht, dass das *endDate*-Element vorhanden sein muss, aber leer sein darf:

```
<xs:element name="endDate" type="xs:date"
  nillable="true" />
```

Im XML-Dokument müssen für den Fall, dass *endDate* einen Nullwert speichert, der XML-Schema-Instanz-Namensraum angegeben und das *endDate*-Element mit *xsi:nil="true"* gekennzeichnet werden:

Listing 6

Schema mit flachen Typdefinitionen.

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- Deklaration eines globalen einfachen unsignedInt-Typs,
    eingeschränkt auf Werte größer null -->
  <xs:simpleType name="unsignedIntGreaterZero">
    <xs:restriction base="xs:unsignedInt">
      <xs:minInclusive value="1"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- Deklaration eines globalen einfachen string-Typs,
    eingeschränkt auf die Werte 'private' und 'business' -->
  <xs:simpleType name="projectType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="private"/>
      <xs:enumeration value="business"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- Globale Deklaration des project-Elements -->
  <xs:element name="project">
    <xs:complexType>
      <xs:all>
        <xs:element name="name" type="xs:string" />
        <xs:element name="startDate" type="xs:date" />
        <xs:element name="endDate" type="xs:date" minOccurs="0" />
        <xs:element name="customerID" type="xs:unsignedInt" minOccurs="0" />
      </xs:all>
      <xs:attribute name="id" type="unsignedIntGreaterZero" use="required" />
      <xs:attribute name="type" type="projectType" use="required" />
    </xs:complexType>
  </xs:element>

  <!-- Das eigentliche Schema -->
  <xs:element name="projects">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="project" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

```

<endDate xsi:nil="true" xmlns:xsi=
  "http://www.w3.org/2001/XMLSchema-
  instance"/>

```

Elemente, die Nullwerte erlauben, erleichtern unter Umständen die spätere Verarbeitung des XML-Dokuments und sind nötig, wenn ein XML-Dokument in ein Objekt mit Nullable-Eigenschaften deserialisiert werden soll. Das *nullable*-Attribut können Sie übrigens nicht nur für einfache Typen, sondern auch für komplexe angeben.

Für Stringelemente besteht häufig die Anforderung, dass diese keine Nullwerte und auch keine Leerwerte erlauben und nicht ausschließlich aus Whitespaces-Zeichen bestehen dürfen. Dieses Ziel erreichen Sie am einfachsten über *xs:token*. Dieser

Typ, für welchen alle Whitespaces-Zeichen durch Leerzeichen ersetzt und am Anfang und am Ende entfernt werden, lässt Strings, die nur aus Whitespaces-Zeichen bestehen, nicht zu. Das *name*-Element im Beispielprojekt sollte etwa mit *xs:token* deklariert werden, um einen mehr oder weniger sinnvollen Namen zu erzwingen.

Sicherstellen von Eindeutigkeit und Integrität von Daten

Zum Sicherstellen von Eindeutigkeit stellt XML Schema gleich mehrere Konstrukte zur Verfügung. Der Datentyp *xs:ID* lässt nur eindeutige Werte zu und ist deswegen bedingt für Schlüsselwerte geeignet. Bedingt deshalb, weil er von *xs:NCName* abgeleitet ist und sein Wert aus diesem Grund ein

gültiger XML-Name sein muss. Ein solcher muss mit einem Buchstaben oder Doppelpunkt beginnen. Zahlwerte oder Werte, die mit einer Zahl beginnen, sind daher prinzipiell nicht möglich. *xs>IDREF* und *xs>IDREFS* sind die Konterparts von *xs:ID*. Diese Typen lassen nur Werte zu, die in einem *xs:ID*-Element beziehungsweise -Attribut enthalten sind. *xs>IDREFS* ist dabei eine Whitespace-begrenzte Liste von *xs:ID*-Werten.

Wesentlich flexibler als *xs:ID* sind die Konstrukte *xs:unique*, *xs:key* und *xs:keyref*. *xs:unique* und *xs:key* bestimmen über XPath-Pfade, welches Element in welchen XML-Knoten eindeutige Werte speichern muss. Der Unterschied ist, dass *xs:unique* in den XML-Knoten auch Nullwerte zulässt, *xs:key* dagegen nicht. *xs:keyref*, das hier nicht näher behandelt wird, schränkt den Wert eines Elements oder Attributs auf einen Wert ein, der über *xs:unique* oder *xs:key* eindeutig ist.

Beide Konstrukte werden direkt unter einem *xs:element*-Element angegeben. Für beide wird in *xs:selector* der XPath-Pfad zu dem Element angegeben, das eindeutig sein soll. Dieser Pfad ist relativ und bezieht sich auf das übergeordnete Element. In einem oder mehreren *xs:field*-Elementen geben Sie den Pfad zu den Attributen oder Elementen an, die eindeutig sein sollen. Für eine Eindeutigkeit über mehrere Attribute/Elemente geben Sie mehrere *xs:field*-Elemente an. Für das *id*-Attribut eines *project*-Elements zeigt Listing 7, wie Eindeutigkeit erreicht wird.

Wichtig ist, dass die Eindeutigkeit immer im Kontext des Elements gilt, in dem *xs:key* oder *xs:unique* angegeben sind. Wird *xs:key* zum Beispiel in einem *order*-Element angebracht und definiert Eindeutigkeit für das *id*-Attribut der dort verwalteten Sequenz von *product*-Elementen, müssen alle *product*-Elemente des jeweiligen *order*-Elements eindeutig sein. Würde *xs:key* allerdings im übergeordneten *orders*-Element angegeben, das eine Sequenz von *order*-Elementen verwaltet, würde sich die Eindeutigkeit auf alle *product*-Elemente in allen *order*-Elementen beziehen. Dieser Fakt kann auch zum Ignorieren von Uneindeutigkeit oder zu falscher Eindeutigkeit führen, wenn Sie *xs:key* oder *xs:unique* im falschen Kontext unterbringen. Besonders beachtenswert ist in diesem Zusammenhang, dass beide Konstrukte ignoriert werden, wenn der Pfad zum Element, das eindeutig sein soll, nicht gültig ist. Mit einem falschen Kontext kann das zu unbemerkten Fehlern führen. Im Beispiel in Listing 7 wä-

re dies der Fall, wenn *xs:key* versehentlich in *project*-Element angelegt wäre. Sie sollten die geplante Eindeutigkeit über *xs:key* und *xs:unique* also immer überprüfen.

Namensräume

Die bisherigen Schemas in diesem Artikel bezogen sich auf XML-Dokumente, die keinem spezifischen Namensraum und damit dem globalen XML-Namensraum zugeordnet sind. Ein XML-Schema kann sich auf diesen globalen und auf einen spezifischen Namensraum beziehen. Wenn das zu überprüfende Dokument oder ein Teil davon einem spezifischen Namensraum zugeordnet ist, müssen Sie diesen im Schema angeben. Aus Platzgründen erläutert der Artikel lediglich die verwendeten Mechanismen. Das Beispielprojekt zum Artikel enthält entsprechende Schemas und XML-Dokumente.

Über das Attribut *targetNamespace* des *xs:schema*-Elements geben Sie den Ziel-Namensraum für das Schema an. Für jedes definierte Attribut und Element wird über den Wert *unqualified* im Attribut *form* bestimmt, dass es dem globalen Namensraum angehört. Der Wert *qualified* gibt an, dass das Attribut oder Element dem spezifischen Namensraum angehört. Der Defaultwert dieses Attributs ist der Wert, der in den Attributen *elementFormDefault* und *attributeFormDefault* des *xs:schema*-Elements angegeben ist. Der Defaultwert dieser Attribute ist *unqualified*.

Wenn das XML-Dokument zum Beispiel über die Angabe eines Namensraums im Wurzelement mit einem Namensraum versehen ist, gilt dieser nur für die Elemente, nicht für die Attribute. Attribute sind dann dem globalen Namensraum zugeordnet. In diesem Fall müssten Sie in *elementFormDefault qualified* und in *attributeFormDefault unqualified* angeben.

Fazit

Dieser Artikel konnte nur die wesentlichen Grundlagen von XML Schema 1.0 behandeln. Nicht angesprochen wurden eine Menge Themen wie Einschränkungen über reguläre Ausdrücke, Ableitungen komplexer Typen, Inhaltsmodelle mit gemischtem und leerem Inhalt, Typsubstitutionen und das Einbetten externer Schemas. Sehr gute weiterführende Informationen finden Sie in [7] und [8]. Die sehr interessanten Erweiterungen von XML Schema 1.1 konnten ebenfalls nicht behandelt werden – einige davon sind allerdings Thema des nächsten Artikels dieser Serie.

Listing 7

Eindeutigkeit über *xs:key*.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="projects">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="project" maxOccurs="unbounded">
          ...
        <xs:key name="id">
          <xs:selector xpath="project" />
          <xs:field xpath="@id" />
        </xs:key>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Zusammengefasst ergibt sich aber, dass XML Schema 1.0 eine starke Schemasprache mit einigen Mankos ist. Die wichtigsten Vorteile sind:

- XML Schema 1.0 ist Quasi-Standard und wird von vielen Tools und Programmiersystemen sowie .NET direkt unterstützt.
- XML Schema 1.0 bietet gute Möglichkeiten, die Struktur eines Dokuments abzubilden und über das starke Typsystem auch den Inhalt festzulegen.
- XML-Schemas können auch direkt in XML-Dokumente integriert werden.

Zusammengefasst sind dies die Nachteile:

- Geschäftsregeln können nur sehr eingeschränkt abgebildet werden, da Kookkurrenz-Einschränkungen und bedingter Inhalt nicht möglich sind. Beides ist allerdings in XML Schema 1.1 enthalten.
- XML Schema ist komplex und schwer zu erlernen. XML-Schemas sind für Menschen nur schwer lesbar.
- Die Fehlermeldungen von XML Schema 1.0 sind für Menschen nicht sehr aussagekräftig. XML Schema 1.1 und Schematron erlauben hingegen – für Assertionen – benutzerdefinierte Fehlermeldungen.
- XML Schema 1.0 legt die Struktur eines XML-Dokuments fest. Offene Schemas sind eingeschränkt und nur über spezielle Techniken wie Namensräume, das Konstrukt *xs:any* oder Typsubstitutionen möglich. XML Schema 1.1 bietet allerdings auch die Möglichkeit, in Schemas unbekannte Elemente zuzulassen. Trotzdem lässt XML Schema nicht alle Möglichkeiten zu, die XML prinzipiell bietet.

Da die gravierenden Nachteile in XML Schema 1.1 größtenteils aufgehoben sind, gäbe es aus Sicht des Autors kaum einen

Grund, auf andere Sprache auszuweichen. Wäre da nicht das Problem, dass XML Schema 1.1 von .NET 4.0 nicht unterstützt werden wird. Dass dies der Fall ist, wurde übrigens auf eine Anfrage über den Microsoft XML Team Blog [9] von Pawel Kadluczka bestätigt. Deshalb müssen Sie zur Implementierung von Geschäftsregeln oder für nicht-deterministische Schemen auf externe Komponenten oder andere Sprachen wie Schematron ausweichen. Das aus der Java-Welt stammende Saxon [10] bietet XML-Schema-1.1-Support, doch leider nur in der kommerziellen Enterprise-Edition. Glücklicherweise können Schematron-Assertionen auch in XML-Schemas integriert werden, sodass die sehr gute Struktur- und Typprüfung von XML Schema 1.0 mit den Assertionen von Schematron kombiniert werden kann. Wie das geht, zeigt dotnetpro im nächsten Teil dieser Serie. **[bl]**

[1] XML Schema Part 1: Structures Second Edition, www.w3.org/TR/xmlschema-1

[2] XML Schema Part 2: Datatypes Second Edition, www.w3.org/TR/xmlschema-2

[3] W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures, www.w3.org/TR/xmlschema11-1

[4] W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, www.w3.org/TR/xmlschema11-2

[5] XML Schema – Data Types Quick Reference, www.dotnetpro.de/SL1004Schematron1

[6] XML Schema – Regular Expressions, www.dotnetpro.de/SL1004Schematron2

[7] Eric van der Vlist, XML Schema, O'Reilly 2002, ISBN 978-0-596-00252-7

[8] XML Schema Tutorial, www.w3schools.com

[9] Microsoft XML Team's WebLog, <http://blogs.msdn.com/xmlteam>

[10] Saxonica, www.saxonica.com