Gebruik van minOccurs=0 of nillable=true

```
<complexType name=”ZoekIngeschrevenPersoonOpGeslachtsnaamFilter”>
        <element name=”geslachtsnaam” type=”CMN:StringFilter” /> [verplicht]
        <element name=”geboortedatum” type=”NP:Geboortedatum” minOccurs=”0” />  [optioneel]
        ...
        <element name=”geslachtsaanduiding” type=”NP:Geslachtsaanduiding” minOccurs=”0” /> [optioneel]
        <element name=”inclusiefNietIngezetenen” type=”boolean” minOccurs=”0” /> [optioneel]
       ...
</complexType>
```

Creeren van een instantie van de gegenereerde .NET class, standaard initialisatie door .NET

```
var filter = new ZoekIngeschrevenPersoonOpGeslachtsnaamFilter();
```

Serialisatie in .NET (WCF) naar xml levert het volgende:

```
<?xml version="1.0"?>
<ZoekIngeschrevenPersonenOpGeslachtsnaamFilter
   xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <geslachtsaanduiding>Man</geslachtsaanduiding> [default waarde: Man. Eerste enum waarde]
   <inclusiefNietIngezetenen>false</inclusiefNietIngezetenen> [default waarde: false. 0 == false]
</ZoekIngeschrevenPersonenOpGeslachtsnaamFilter>
```

Serialisatie met JAXB naar xml levert het volgende:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<zoekIngeschrevenPersonenOpGeslachtsnaamFilter
     xmlns=http://www.stufstandaarden.nl/koppelvlak/STUF4/Berichten/IngeschrevenPersoon
     xmlns:ns2="http://www.stufstandaarden.nl/koppelvlak/STUF4/Entiteiten/NatuurlijkPersoon">
  <inclusiefNietIngezetenen>false</inclusiefNietIngezetenen>
</zoekIngeschrevenPersonenOpGeslachtsnaamFilter>
```

```
<complexType name=”ZoekIngeschrevenPersoonOpGeslachtsnaamFilter”>
        <element name=”geslachtsnaam” type=”CMN:StringFilter” /> [verplicht]
        <element name=”geboortedatum” type=”NP:Geboortedatum” minOccurs=”0” />  [optioneel]
        ...
        <element name=”geslachtsaanduiding” type=”NP:Geslachtsaanduiding” nillable=”true” /> [optioneel]
        <element name=”inclusiefNietIngezetenen” type=”boolean” /> [verplicht]
       ...
</complexType>
```

Creeren van een instantie van de gegenereerde .NET class, standaard initialisatie door .NET

```
var filter = new ZoekIngeschrevenPersoonOpGeslachtsnaamFilter();
```

Serialisatie in .NET (WCF) naar xml levert het volgende:

```
<?xml version="1.0"?>
<ZoekIngeschrevenPersonenOpGeslachtsnaamFilter
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <geslachtsaanduiding xsi:nil="true" /> [default waarde: null]
    <inclusiefNietIngezetenen>false</inclusiefNietIngezetenen>
</ZoekIngeschrevenPersonenOpGeslachtsnaamFilter>
```

Serialisatie met JAXB naar xml levert het volgende:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<zoekIngeschrevenPersonenOpGeslachtsnaamFilter
      xmlns=http://www.stufstandaarden.nl/koppelvlak/STUF4/Berichten/IngeschrevenPersoon
      xmlns:ns2="http://www.stufstandaarden.nl/koppelvlak/STUF4/Entiteiten/NatuurlijkPersoon">
   <geslachtsaanduiding xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:nil="true"/>
   <inclusiefNietIngezetenen>false</inclusiefNietIngezetenen>
</zoekIngeschrevenPersonenOpGeslachtsnaamFilter>
```

Overeenkomsten in serialisatie tussen JAXB en .NET:

- Elementen van complexType types met attribuut minOccurs="0" worden bij 'null' waarde niet geserialiseerd naar XML. In dit voorbeeld is dit het geval bij de geboortedatum element
- Elementen van boolean type met attribuut minOccurs="0" krijgt bij serialisatie de default value. In het geval van een boolean is dit dus false. Dit zal (hoogst waarschijnlijk, maar niet getest) ook het geval zijn bij nummer (int, decimal, long) en datum types

Verschillen in serialisatie tussen JAXB en .NET:

- Elementen van enum types met attribuut minOccurs="0" krijgt in .NET bij serialisatie de waarde van de eerste enum waarde, in dit geval 'Man'. In Java (JAXB) is een enum type nillable. Hierdoor is het in Java wel mogelijk om geen waarde toe te kennen aan een enum type en wordt deze bij 'null' waarde niet geserialiseerd naar xml

Conclusie:

- Gebruik bij elementen van struct types (bool, int, long, dateTime) nillable="true" om aan te geven dat het element optioneel is.
- Gebruik bij elementen van enum types ook nillable="true" om aan te geven dat het element optioneel is. Dit zorgt voor dezelfde interpretatie bij .NET en Java consumers/providers
- Als een element optioneel moet zijn, maar de default waarde is de gewenste waarde bij optionaliteit, dan moet dit element gewoon verplicht worden gemaakt. Dit maakt de schema o.a. "minder complex" en draagt bij aan de eenduidigheid

Gebruik van abstract complexTypes

In de huidige StUF wordt er gebruik gemaakt choice constructs om aan te geven dat een element van een aantal types kan zijn. Bijvoorbeeld: een verblijfsadres kan een nummeraanduiding of een adresbuitenland zijn. Een ander voorbeeld is een rechtspersoon. Een ingeschreven natuurlijk persoon of ingeschreven niet natuurlijk persoon kunnen eigenaar zijn van een maatschappelijke activiteit

```
<complexType name="Nummeraanduiding">
 <sequence></sequence>
</complexType>
<complexType name="AdresBuitenland">
 <sequence></sequence>
</complexType>
<complexType name="VerblijfsAdres">
 <sequence>
  <choice>
   <element name="nummeraanduiding" type="poc:Nummeraanduiding" />
   <element name="adresBuitenland" type="poc:AdresBuitenland" />
  </choice>
 </sequence>
</complexType>
```

In de .NET documentatie(*) is aangegeven dat een choice niet mag worden gebruikt in een schema. Wordt het toch gebruikt, dan wordt een oudere versie van code generatie gebruikt en wordt de choice construct gemapt naar System.Xml.XmlNode[].
Binnen Java wordt, volgens IBM documentatie(**), de choice construct door de JAX-RPC code generator gemapt naar javax.xml.soap.SOAPElement. SOAPElement.
Voor zowel .NET als Java is het hierdoor niet meer mogelijk om vanuit code af te leiden welke types aan zo'n XmlNode[] c.q. SOAPElement mag worden toegekend.

In hetzelfde document van IBM wordt polymorfisme als alternatief geboden voor de choice construct. In de POC is polymorfisme ook gebruikt als alternatief voor choices. Bovengenoemd voorbeeld kan door toepassen van polymorfisme als volgt worden gemoduleerd:

```
<complexType name="VerblijfsAdres" abstract="true"><sequence /></complexType>
<complexType name="Nummeraanduiding">
  <complexContent>
    <extension base="VerblijfsAdres />
      <sequence>
        ...
      </sequence>
    </extension>
  </complexContent>
</complexType>
<complexType name="AdresBuitenland">
  <complexContent>
    <extension base="VerblijfsAdres />
      <sequence>
        ...
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Kanttekening.
De polymorfisme constructie kan niet as-is worden toegepast omdat de .NET code generator (svcutil) de attribuut abstract=”true” niet ondersteunt. Wordt het toch gebruikt, dan valt de .NET code generator terug op de legacy manier van genereren, waardoor er veel class duplicatie ontstaat in de gegenereerde code.
Op dit moment is er voor gekozen om de abstract=”true” attribuut te verwijderen voor de abstract complexTypes. Dit heeft als nadeel dat de als abstract gemarkeerde classes niet meer abstract zijn, waardoor het mogelijk is om een instantie van de abstract class (v.b. VerblijfsAdres) te creeren. Om het inzichtelijk te maken dat deze classes abstract zijn, is aan deze classes Abstract als prefix aan de naam toegevoegd.

* .NET Data Contract Schema Reference
** Web services tip: Use polymorphism as an alternative to xsd:choice

Gebruiken/conformeren aan wrapped-document/literal style

De wrapped-document/literal style is de meest geaccepteerde style voor het definieren van webservice interfaces. Webservices die aan deze style voldoen zijn daardoor highly interoperable. De huidige code generatoren in .NET en Java ondersteunen deze style en mappen een wrapped-document/literal compliant service operatie automatisch naar een methode van een class.

De wrapped-document/literal pattern bevat de volgende afspraken waaraan de WSDL moet voldoen:

- Elk bericht (zowel input als output) bevat slechts één body part
- Een body part heeft als naam “parameters” en verwijst naar één element. Dit ‘wrapper’ element bevat de parameters van de operatie
- Elk parameter van een object-type (string, dateTime, complexType) bevat de nillable=”true” attribuut.
  Kanttekening. Het is (nog) niet uitgezocht of dit een .NET specifieke requirement is
- De naam van een input wrapper element komt overeen met de naam van de operatie
- De naam van een output wrapper element komt overeen met de naam van de operatie + Response
- De soap:binding element bevat de style=”document” attribuut
- Elk soap:body element bevat de use=”literal” attribuut

Bronnen:
Usage of document/literal wrapped pattern in WSDL design
Creating doc-lit WSDLs that “unwrap” nicely