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>
```

Creëren 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 verplicht worden gemaakt. Dit maakt het schema o.a. "minder complex" en draagt bij aan de eenduidigheid.
- Gebruik minOccurs="0" bij elementen van een complex type om aan te geven dat het element optioneel is.
- Gebruik nooit zowel minOccurs="0" als nillable="true" om aan te geven dat een element optioneel is. Dit biedt geen meerwaarde boven minOccurs="0" of nillable="true".

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>
<complexType name="VerblijfsAdresRelatie">
        <complexContent>
                <extension base="CMN:Relatie">
                        <sequence>
                                <element name="adres" type="CMN:VerblijfsAdres" />
                        </sequence>
                </extension>
        </complexContent>
</complexType>
```

Deze constructie maakt het mogelijk om een element van het Nummeraanduiding of AdresBuitenland toe te kennen aan het adres element van het VerblijfsAdresRelatie type. Het is niet mogelijk om een element van een andere type toe te kennen aan het adres element.
Een door .NET/Java geserialiseerde VerblijfsAdresRelatie met respectievelijk een Nummeraanduiding en een AdresBuitenland als adres type naar XML ziet er dan als volgt uit:

```
<VerblijfsAdresRelatie xmlns=\"http://www.stufstandaarden.nl/koppelvlak/STUF4/Entiteiten/Gemeenschappelijk\"
xmlns:i=\"http://www.w3.org/2001/XMLSchema-instance\">
  <periodeGeldigheid>
  ...
  </periodeGeldigheid>
  <adres i:type="Nummeraanduiding">
  ...
  </adres>
</VerblijfsAdresRelatie>

<VerblijfsAdresRelatie xmlns=\"http://www.stufstandaarden.nl/koppelvlak/STUF4/Entiteiten/Gemeenschappelijk\"
xmlns:i=\"http://www.w3.org/2001/XMLSchema-instance\">
  <periodeGeldigheid>
  ...
  </periodeGeldigheid>
  <adres i:type="AdresBuitenland">
  ...
  </adres>
</VerblijfsAdresRelatie>
```

Door het toevoegen van de type attribuut aan het adres element, kan een consumer van de XML deze weer correct interpreteren.

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 het 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 creëren. Om 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:

1. Elk bericht (zowel input als output) bevat slechts één body element
2. De naam van het body element is “parameters” en verwijst naar één ‘wrapper’ element. Dit ‘wrapper’ element bevat de parameters van de operatie
3. De naam van een input wrapper element komt overeen met de naam van de operatie
4. De naam van een output wrapper element komt overeen met de naam van de operatie + Response
5. De soap:binding element bevat de style=”document” attribuut
6. Elk soap:body element bevat de use=”literal” attribuut
7. Elk parameter van een object-type (string, dateTime, complexType) bevat de nillable=”true” attribuut. Dit is een .NET specifieke requirement. De code generator zal de wrapper element niet ‘un-wrappen’ als aan deze requirement niet voldaan. Dit punt zal verderop in een apart paragraaf worden uitgewerkt.

In onderstaand tabel is als voorbeeld de **ZoekIngeschrevenPersonenOpGeslachtsnaam** operatie van de **BevraagIngeschrevenPersoon** service uitgewerkt.

De definitie van de **BevraagIngeschrevenPersoon** service staat in **IngeschrevenPersoon.wsdl** bestand en de bijbehorende bericht definities staan in **StUF4_msg_ingeschrevenpersoon.xsd**

IngeschrevenPersoon.wsdl (bevat de definitie van de SOAP service interface)

```
<definitions ...>
...
  <!-- 1. Input bericht bevat slechts één body element -->
  <message name=”ZoekIngeschrevenPersonenOpGeslachtsnaamIn”>
    <!-- 2. naam body (wrapper) element is parameters -->
    <!-- 3. naam ‘wrapper’ element komt overeen met de naam van de operatie:
            ZoekIngeschrevenPersonenOpGeslachtsnaam -->
    <part name=”parameters” element=”MIP:ZoekIngeschrevenPersonenOpGeslachtsnaam” />
  </message>
  <!-- 1. Input bericht bevat slechts één body element -->
  <message name=”ZoekIngeschrevenPersonenOpGeslachtsnaamOut”>
    <!-- 2. naam body (wrapper) element is parameters -->
    <!-- 4. naam ‘wrapper’ element komt overeen met de naam van de operatie + Response:
            ZoekIngeschrevenPersonenOpGeslachtsnaamResponse -->
    <part name=”parameters” element=”MIP:ZoekIngeschrevenPersonenOpGeslachtsnaamResponse” />
  </message>
  ...
  <binding name=”SOAPBevraagIngeschrevenPersoon” type=”MIP:BevraagIngeschrevenPersoon”>
     <!—5. Soap:binding element bevat de style=”document” attribuut
     <soap:binding style=”document” ... />
     <operation name=”ZoekIngeschrevenPersoonOpGeslachtsnaam”>
```

```
      <soap:operation soapAction="" />
      <input>
        <!-- 6. Elk soap:body element bevat de use="literal" attribuut
        <soap:body use="literal" />
      </input>
      <output>
        <!-- 6. Elk soap:body element bevat de use="literal" attribuut
        <soap:body use="literal" />
      </output>
    </operation>
    ...
  </binding>
  ...
</definitions>
```

StUF4_msg_ingeschrevenpersoon.xsd (bevat de definities van de request en response berichten)

```
<schema ...>
  ...
   <!-- 3. naam 'wrapper' element komt overeen met de naam van de operatie:
           ZoekIngeschrevenPersonenOpGeslachtsnaam -->
  <element name="ZoekIngeschrevenPersonenOpGeslachtsnaam">
    <complexType>
      <sequence>
        <!-- 7. object-type element moeten de nillable="true" attribuut hebben om in .NET de wrapper element correct te
                laten un-wrappen -->
        <element name="filter" type="MIP:ZoekIngeschrevenPersonenOpGeslachtsnaamFilter" nillable="true" />
      </sequence>
    </complexType>
  </element>
    <!-- 4. naam 'wrapper' element komt overeen met de naam van de operatie + Response:
            ZoekIngeschrevenPersonenOpGeslachtsnaamResponse -->
  <element name="ZoekIngeschrevenPersonenOpGeslachtsnaamResponse">
    <complexType>
      <sequence>
        <!-- 7. object-type element moeten de nillable="true" attribuut hebben om in .NET de wrapper element correct te
                laten un-wrappen -->
        <element name="ingeschrevenPersonen" type="MIP:IngeschrevenPersonenBeperkt" nillable="true" />
      </sequence>
    </complexType>
  </element>
  ...
</schema>
```

Toekennen van de nillable="true" attribuut aan object-type elementen van de 'wrapper' element

Wanneer een service definitie zich conformeert aan de wrapped document/literal pattern, dan kunnen de code generatoren (.NET WCF en Apache Axis2 > v1.1.1) worden geconfigureerd om bij het genereren van code te un-wrappen, wat betekent dat een service operatie met bijbehorende parameters wordt gemapt naar een methode en parameters met dezelfde naam. Voor de als voorbeeld gebruikte ZoekIngeschrevenPersonenOpGeslachtsnaam operatie, zal de .NET code generator (svcutil), de volgende interface en methode genereren:

```
public interface BevraagIngeschrevenPersoon
{
```

```
  ...
  IngeschrevenPersonenBeperkt ZoekIngeschrevenPersonenOpGeslachtsnaam(
    ZoekIngeschrevenPersonenOpGeslachtsnaamFilter filter);
  ...
}
```

Wordt er niet aan alle afspraken voldaan, dan zal de .NET code generator ervoor kiezen om ook voor de 'wrapper' elementen classes te genereren. De gegenereerde code voor de ZoekIngeschrevenPersonenOpGeslachtsnaam operatie ziet er dan als volgt uit:

```
public interface BevraagIngeschrevenPersoon
{
  ...
  ZoekIngeschrevenPersonenOpGeslachtsnaamResponse ZoekIngeschrevenPersonenOpGeslachtsnaam(
    ZoekIngeschrevenPersonenOpGeslachtsnaamRequest request);
  ...
}
public partial class ZoekIngeschrevenPersonenOpGeslachtsnaamRequest
{
  public ZoekIngeschrevenPersonenOpGeslachtsnaamRequestBody Body;
}
public partial class ZoekIngeschrevenPersonenOpGeslachtsnaamRequestBody
{
  public ZoekIngeschrevenPersonenOpGeslachtsnaamFilter filter;
}
public partial class ZoekIngeschrevenPersonenOpGeslachtsnaamResponse
{
  public ZoekIngeschrevenPersonenOpGeslachtsnaamResponseBody Body;
}
public partial class ZoekIngeschrevenPersonenOpGeslachtsnaamResponseBody
{
  public IngeschrevenPersonenBeperkt ingeschrevenPersonen;
}
```

In de code voorbeelden hierboven is te zien dat wanneer de code generatoren niet un-wrapt, omdat niet aan alle 'wrapped document/literal' voorwaarden zijn voldaan, dat er minimaal 4 extra classes zijn gegenereerd en dat de input en output parameters middels deze 4 classes worden doorgegeven. Dit maakt de code complexer in gebruik.

Voorwaarde 7, alle object-type elementen moeten de nillable="true" attribuut hebben, kan voor verwarring zorgen. Verplichte object-type elementen moeten namelijk ook de nillable="true" attribuut hebben.
Echter, deze voorwaarde geldt alleen voor de object-type elementen van wrapper elementen en door de parameters (in dit voorbeeld zoek filter parameters) te encapsuleren in een complexType (in dit geval ZoekIngeschrevenPersonenOpGeslachtsnaamFilter) is er maar één element waaraan de nillable="true" attribuut moet worden toegekend. Indien er meerdere output parameters zijn, kunnen deze ook worden geëncapsuleerd in één complexType. Hiermee wordt de semantiek van de operaties en bijbehorende bericht types minimaal aangepast.
Ook is de impact voor Java consumers en providers nihil. Zoals aangegeven in de bron 'Creating doc-lit WSDLs that "unwrap" nicely' ondersteunt de Apache Axis2 WSDL2java code generator ook de unwrapping mode door gebruik van de **-uw** command line parameter.

Bronnen:

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

Versioning van service contracten

Voor het versionen van service contracten moet er rekening worden met de volgende wijzigingen die kunnen plaatsvinden in een service contract:

1. Breaking/Non backward-compatible changes
2. Non-breaking/backward-compatible changes

Bij een breaking change kunnen consumers en providers niet meer correct functioneren als de change niet wordt doorgevoerd.

Voorbeelden van breaking changes:

3. het veranderen van de minOccurs attribuut van een element van 0 (optioneel) naar 1 (verplicht)
4. het veranderen van de type van een element
5. het wijzigen van de (target) namespace van de xsd

Voorbeelden van non-breaking changes:

6. het toevoegen van een nieuwe operatie/method
7. het veranderen van de minOccurs attribuut van een element van 1 (verplicht) naar 0 (optioneel)

Er zijn op dit moment twee veel gebruikte versioning strategiën:

8. Een nummer in de volgende formaat: [major nummer].[minor nummer]
   Optioneel kan dit worden uitgebreid met een revision nummer en eventueel een increment nummer tot de volgende formaat: [major].[minor].[revision].[increment]
9. Een datum om aan te geven wanneer een versie is vrijgegeven

Voorbeelden van het gebruik van de twee versioning strategiën:

http://www.stufstandaarden.nl/koppelvlak/STUF4/Entiteiten/Gemeenschappelijk/v4.1
http://www.stufstandaarden.nl/koppelvlak/STUF4/Entiteiten/Gemeenschappelijk/v4/1
http://www.stufstandaarden.nl/koppelvlak/STUF4/Entiteiten/Gemeenschappelijk/4/1
http://www.stufstandaarden.nl/koppelvlak/STUF4/Entiteiten/Gemeenschappelijk/2017/7/1

Het gebruiken van een versie nummer heeft als voordeel dat het ophogen van één van de nummers kan worden gekoppeld aan een soort change, bij het doorvoeren van één of meerdere breaking changes wordt de major nummer opgehoogd en bij het doorvoeren van één of meerdere non-breaking changes wordt de minor version opgehoogd.

Bij het gebruiken van een datum is het niet mogelijk om dezelfde versioning strategie als bij een nummer versie toe te passen. De datum versioning strategie wordt daarom ook voornamelijk gebruikt om een major release (een release met breaking changes) te identificeren. Voor non-breaking changes moet dan een andere strategie worden gekozen. In de literatuur is op het moment van dit document geen best practice gevonden hoe minor releases moet worden geïdentificeerd.

Voorkeur versie strategie

De versie nummer strategie heeft daarom de voorkeur gekregen als versioning strategie voor StUF4 en is/wordt als volgt geïmplementeerd:

10. De major nummer wordt opgenomen in de namespace van de schema's, v.b.:
    http://www.stufstandaarden.nl/koppelvlak/STUF4/Entiteiten/Gemeenschappelijk/v4
    Bij een breaking change wordt de namespace gewijzigd, wat er automatisch voor zorgt dat consumers en providers de changes moet doorvoeren om de versie te kunnen ondersteunen.
11. Voor de minor nummer en indien nodig/gewenst de revision en increment nummers wordt de version attribuut van de xsd:schema element gebruikt, v.b.:
    <schema version="v4.0.1">....</schema>
    Hiermee kunnen de bouwers van consumers en providers in de schema's zien bij welke versie de xsd's horen.

Well Known Text (WKT)

Well Known Text (WKT) is een Open Geospatial Consortium (OGC) standaard dat wordt gebruikt om geometrie data te representeren in een tekstuele formaat. De voor RSGB bevragingen benodigde geometrie typen, zoals punt, lijn, multilijn, polygoon, multipolygoon en geometrie-collectie worden door deze standaard ondersteund. Dit houdt in dat de XML-schema's geen afhankelijkheid krijgen naar externe schema's en dat alle codegeneratie-tools ermee overweg kunnen. Alle geo-tools kunnen met dit formaat overweg, waardoor providers en consumers van zowel de XML-SOAP als de JSON varianten, zonder programmeren gebruik kunnen maken van dezelfde geometrie-standaard.

De WKT-standaard is opgenomen in ISO/IEC 13249-3:2016 standard, "Information technology – Database languages – SQL multimedia and application packages – Part 3: Spatial" (SQL/MM) and ISO 19162:2015, "Geographic information – Well-known text representation of coordinate reference systems"

De WellKnownText complexType is gedefinieerd in de StUF4_ent_extern_geometrie.xsd.

JSON API

Voor de JSON-API wordt gebruik gemaakt van dezelfde schemadefinities als de SOAP variant. De naam van de JSON-services zijn gelijk aan de messages in de WSDL's. Alle propertynamen en bijbehorende datatypes zijn conform de schedefinities.

Om de aanroep middels URL's makkelijker te maken zijn de vraagberichten platgeslagen. Bij zoekvragen waar een periode in is opgenomen, worden 'van' en 'totenmet' in de url opgenomen, in plaats van een periode-object.