XML Schema: sprievodca nielen pre tvorcov SOAP služieb

Table of Contents

Čo je XML Schema?

Formát XML je dlhoročný štandard na uchovávanie a prenos štruktúrovaných dát. Je textový, je univerzálny, je primerane čitateľný nielen strojom, ale aj človekom, a spája sa s ním množstvo technológií, napríklad **XHTML** (jazyk pre webové stránky), **XSLT** (jazyk na transformácie XML do XML, či iných formátov), či **SOAP** (webové služby).

XML je "jazyk na tvorbu jazykov", a jednotlivé dokumenty musia spĺňať konkrétne pravidlá:

- *"Ktoré elementy sú povolené na tomto mieste?* môžem v elemente `<b>` mať tabuľkový element `<table>`? Ak nie, ktoré elementy tam môžu byť?
- *V akom poradí musia ísť konkrétne elementy?* Som povinný mať najprv meno a potom priezvisko, alebo môžu ísť v ľubovoľnom poradí?
- *Aký dátový typ má obsah elementu?* Je to číslo? Dátum? Zložený element?
- *Aké sú pokročilé pravidlá, napríklad pre jedinečnosť?* Koľkokrát sa v texte môže vyskytnúť element s identifikátorom `2435`?

XML Schema je jazyk, ktorý určuje pravidlá, ktoré musia spĺňať jednotlivé elementy dokumentu XML. Inými slovami, definuje *gramatiku* pre XML dokumenty.

Kde sa používa XML Schema?

XML Schema sa používa na viacerých miestach:

- **SOAP** služby definujú štruktúru správ pomocou XML schémy.
- Java technológia **JAXB** umožňuje konvertovať XML na objekty a späť. *XML Schema* určuje pravidlá prevodu.
- vývojárske prostredia pre prácu XML umožňujú použiť *XML Schemu* na automatické dopĺňanie (*autocomplete*) povolených elementov

Minimalistická schéma

XML schéma sa píše v jazyku XML. To je síce zvláštne, ale umožňuje to mať jednotný formát pre dokumenty i ich gramatiky.

Minimalistická schéma, ba priam jej kostra, vyzerajú nasledovne:

```
<?xml version="1.0"?>

<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar">
</schema>
```

- Schéma má koreňový element `<schema>`.
- Všetky elementy jazyka XML schémy patria do menného priestoru `http://www.w3.org/2001/XMLSchema`.
- Atribút `targetNamespace` hovorí, do ktorého menného priestoru budú patriť elementy dokumentu popisovaného schémou.

Varianty minimalistickej schémy

V tomto prípade sme element `schema` zaviedli do implicitného menného priestoru (*default namespace*). V niektorých príkladoch sa uvádza explicitný prefix menného priestoru a elementy schémy majú dohodnutý prefix `xsd:`, či `xs:`.

```
<?xml version="1.0"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="urn:example:calendar">
</xsd:schema>
```

Jednoduchá schéma

Predstavme si jednoduchý bežný document:

```
<calendar xmlns="urn:example:calendar">
    3 events
</calendar>
```

Dokument má jediný (koreňový) element `<calendar>`, ktorého obsahom je reťazec.

Element navyše patrí do menného priestoru (*namespace*) `urn:example:calendar`.

Schéma, ktorá popisuje uvedený dokument, vyzerá nasledovne:

```
<?xml version="1.0"?>

<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar">
    <element name="calendar" type="string" />
</schema>
```

Schéma popisuje dokumenty s jediným koreňovým elementom `<calendar>`, ktorý má povolený len reťazcový obsah, bez elementov. Reťazce sú indikované dátovým typom v atribúte `type`, kde `string` reprezentuje *reťazce* v jazyku XML schémy.

XML Schema má zabudovaných štyridsaťštyri (!) jednoduchých dátových typov. Medzi najbežnejšie patrí `string` (reťazce), `double` (desatinné čísla), `int` pre celé čísla, či `dateTime` pre dátumy a časy. Úplný zoznam je v špecifikácii, konkrétne zozname primitívnych dátových typov a v dodatočnom zozname odvodených dátových typov.

Validácia XML schémy

Na validáciu XML schémy môžeme použiť viacero nástrojov, napríklad `xmllint`:

```
xmllint calendar.xml --schema calendar.xsd
```

Dokument, ktorý spĺňa všetky pravidlá danej XML schémy, je **validný**. Hovorievame tiež, že takýto dokument je **inštanciou** príslušnej XML schémy.

Schéma s vnorenými elementami

Vylepšime dokument o kalendár, ktorý obsahuje vnorené elementy:

```
<calendar xmlns="urn:example:calendar">
    <event>Conference Intro at 17:00</event>
    <event>On XML Schemas at 20:00</event>
    <event>Conference outro</event>
</calendar>
```

Schéma má pravidlá:

1. Koreňový element je `<calendar>`.
2. Kalendár obsahuje jeden a viac elementov `<event>` pre konkrétne udalosti.
3. Každá udalosti obsahuje len text.

Prvý nástrel schémy bude vyzerať nasledovne:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:example:calendar">
    <element name="calendar"> ①
        <complexType> ②
            <sequence> ③
                <element name="event" maxOccurs="unbounded" type="string" /> ④
            </sequence>
        </complexType>
    </element>
</schema>
```

① Element `<calendar>` v schéme je jediný povolený koreňový element. (Pravidlo 1.)

② Element `<complexType>` hovorí, že `<calendar>` bude obsahovať podelementy. Ide o **komplexný typ**, na rozdiel od predošlej verzie, ktorá bola *simple type*, jednoduchý typ.

③ Element `<sequence>` hovorí, že elementy v kalendári musia ísť v takom poradí, v akom sú uvedené v schéme. V tomto prípade to nezaváži, pretože `<calendar>` obsahuje výhradne elementy rovnakého typu `<event>`, kde na poradí aj tak nezáleží.

④ Element `<event>` je jednoduchý element, ktorý obsahuje reťazce (typ `string`). Element `maxOccurs` nastavuje neobmedzený počet opakovaní, teda neobmedzený počet udalostí v kalendári. Minimálny počet opakovaní je jedna, čo možno voliteľne nastaviť v elemente `minOccurs`.

Pokus o validáciu XML dokumentu oproti uvedenej schéme zlyhá:

```
Schemas validity error : Element '{urn:example:calendar}event': This element is not
expected. Expected is ( event ).
```

Typický problém: kvalifikované elementy a lokálne elementy

XML Schema má v štandardnom nastavení zvláštne správanie:

XML schéma v štandardnom správaní hovorí, že *žiadny z lokálnych elementov nesmie byť kvalifikovaný*.

Globálny element (*global element*) je taký, ktorý je priamym potomkom elementu `<schema>` v XML schéme. Všetky ostatné elementy deklarované v schéme sú **lokálne**.

Example 1. Príklady globálnych a lokálnych elementov

V našej schéme máme jeden globálny element `<calendar>` a jeden lokálny element `<event>`.

Kvalifikovaný element (*qualified element*) patrí do nejakého menného priestoru.

Example 2. Príklady kvalifikovaných elementov

V našom XML dokumente (inštancii, nie schéme!) sme povedali, že všetky elementy patria do menného priestoru `urn:example:calendar`, ktorý sme zároveň vyhlásili za implicitný.

Chybová hláška hovorí o porušení pravidla XML schémy. Element `<event>`, ktorý je v schéme lokálny, je v XML inštancii kvalifikovaný (patrí do menného priestoru), čo nie je povolené.

Validátor jasne hovorí, že element s kvalifikovaným menom `{urn:example:calendar}event` v elemente kalendára nie je povolený. Namiesto neho je očakávaný nekvalifikovaný element `event`, ktorý nepatrí do žiadneho menného priestoru.

Ako z toho von?

Možnosti sú dve:

1. Buď upravíme XML schému a zrušíme pravidlo o nekvalifikovaných lokálnych elementoch.
2. Alebo upravíme dokument tak, aby zodpovedal schéme a "odkvalifikujeme" elementy udalostí. Toto však robiť nebudeme, lebo je to proces, ktorý je zložitý a v našom prípade nedáva veľký zmysel.

Zapnutie podpory pre kvalifikované lokálne elementy: `elementFormDefault`

```
<?xml version="1.0"?>

<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar"
        elementFormDefault="qualified"> ①
    <element name="calendar">
        <complexType>
            <sequence>
                <element name="event" maxOccurs="unbounded" type="string" />
            </sequence>
        </complexType>
    </element>
</schema>
```

① Atribút `elementFormDefault` s hodnotou `qualified` hovorí, že všetky lokálne elementy musia byť kvalifikované. To je zmena oproti implicitnej hodnote `unqualified`, ktorú sme mali v predošlých príkladoch.

Teraz už validácia prejde.

Nezabudnime, že v inštancii sú všetky elementy kvalifikované, pretože patria do menného priestoru `urn:example:calendar` (podľa zásady o dedení menných priestorov v XML), ktorý je navyše implicitný:

```
<calendar xmlns="urn:example:calendar">
    <event>Conference Intro at 17:00</event>
    <event>On XML Schemas at 20:00</event>
    <event>Conference outro</event>
</calendar>
```

Táto inštancia je ekvivalentná explicitnej verzii, kde každý element vyfasuje explicitný prefix.

```
<cal:calendar xmlns:cal="urn:example:calendar">
    <cal:event>Conference Intro at 17:00</cal:event>
    <cal:event>On XML Schemas at 20:00</cal:event>
    <cal:event>Conference outro</cal:event>
</cal:calendar>
```

Schéma s komplexnými vnorenými elementami

Predstavme si teraz ešte zložitejšiu inštanciu:

```
<calendar xmlns="urn:example:calendar">
    <event>
        <date>2019-05-30T09:00:00</date>
        <description>Welcome Drink</description>
    </event>
</calendar>
```

XML schéma, ktorá popisuje tento dokument:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:example:calendar" elementFormDefault="qualified"> ①
    <element name="calendar">
        <complexType>
            <sequence>
                <element name="event" minOccurs="0" maxOccurs="unbounded"> ②
                    <complexType>
                        <sequence>
                            <element name="date" type="dateTime" /> ③
                            <element name="description" type="string" /> ④
                        </sequence>
                    </complexType>
                </element>
            </sequence>
        </complexType>
    </element>
</schema>
```

① Schéma už rovno zapína pravidlo o povinnosti kvalifikovať všetky elementy, aj globálne, aj lokálne.

② Lokálny element `<event>` je po novom *komplexným*, pretože môže obsahovať dva podelementy pre dátum (`<date>`) a popis (`<description>`). Sekvencia hovorí, že elementy musia ísť v presnom poradí, najprv dátum a potom popis, pričom výmena nie je povolená.

③ Element `<date>` pre dátum má dátový typ `dateTime`, čo je zabudovaný dátový typ pre dátumy a časy.

④ Element `<description>` pre popis je reťazcový.

Keďže elementov typu `<event>` môže byť nula až nekonečno, povolené sú aj extrémne varianty.

Prázdny kalendár:

```xml
<calendar xmlns="urn:example:calendar" />
```

Kalendár s dvoma udalosťami.

```xml
<?xml version="1.0"?>

<calendar xmlns="urn:example:calendar">
    <event>
        <date>2019-05-30T09:00:00</date>
        <description>Welcome Drink</description>
    </event>
    <event>
        <date>2019-05-30T10:00:00</date>
        <description>Conference</description>
    </event>
</calendar>
```

Schéma s viacnásobne vnorenými elementami

Elementy môžeme vnárať aj viacnásobne. Pridajme ku každej udalosti aj zoznam účastníkov.

```xml
<calendar xmlns="urn:example:calendar">
    <event>
        <date>2019-05-30T09:00:00</date>
        <description>Welcome Drink</description>
        <participants> ①
            <participant>John Doe</participant> ②
            <participant>Jane Doe</participant> ②
        </participants>
    </event>
    <event>
        <date>2019-05-30T10:00:00</date>
        <description>Conference</description> ③
    </event>
</calendar>
```

① Všimnime si, že prvá udalosť má dvoch účastníkov uvedených v rámci elementu `<participants>`.

② Každý účastník má svoj vlastný element, kde uvedieme jeho meno.

③ Udalosť nemusí mať žiadnych potvrdených účastníkov.

Schéma následne zopakuje trik s vnáraním elementov:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar"
        elementFormDefault="qualified">
    <element name="calendar">
        <complexType>
            <sequence>
                <element name="event" minOccurs="0" maxOccurs="unbounded">
                    <complexType>
                        <sequence>
                            <element name="date" type="dateTime" />
                            <element name="description" type="string" />
                            <element name="participants" minOccurs="0"> ①
                                <complexType>
                                    <sequence>
                                        <element name="participant" type="string"
maxOccurs="unbounded" /> ②
                                    </sequence>
                                </complexType>
                            </element>
                        </sequence>
                    </complexType>
                </element>
            </sequence>
        </complexType>
    </element>
</schema>
```

① Pribudol jeden lokálny element `<participants>`, ktorý má minimálny počet výskytov nastavený na nulu, čo je ekvivalent nepovinného elementu. Ide o komplexný element so sekvenciou vnorených elementov rovnakého typu.

② Každý účastník je reťazcového typu.

Štýl matrioška

V schéme máme jeden globálny element `<calendar>` a viacero lokálnych elementov: `<event>`, v ňom `<date>`, `<description>` a `<participants>`, a v rámci neho účastníka `<participant>`.

Tento štýl vnárania elementov sa niekde nazýva **matrioška** podľa slávnej ruskej bábiky, ktorá obsahuje bábiky, ktoré obsahujú bábiky.

Schéma s viacerými koreňovými elementami

Globálne elementy schémy určujú povolené koreňové elementy. Doposiaľ sme mali povolený jediný koreňový element `<calendar>`, ale sú situácie, keď jedna schéma popisuje viacero možných inštancií s rozličnými koreňmi.

Medzi príklady z praxe patrí:

- jazyk **DocBook** pre písanie dokumentácie, ktorý povoľuje knihy `<book>`, ale aj články `<article>`
- formát správ vo webových službách **SOAP**, ktorý povoľuje vlastnú definíciu správ pre požiadavky a odpovede. Príkladom môže byť `<CalendarResponse>` pre odpoveď a `<CalendarRequest>` pre požiadavku.

Pridajme si do schémy ďalší koreňový element pre dokument reprezentujúci jednu udalosť.

```
<event xmlns="urn:example:calendar">
    <date>2019-05-30T09:00:00</date>
    <description>Welcome Drink</description>
    <participants>
        <participant>John Doe</participant>
        <participant>Jane Doe</participant>
    </participants>
</event>
```

Schéma, ktorá zvládne aj kalendár, aj jednu udalosť vyzerá nasledovne. Nie je to vonkoncom optimálna schéma, pretože sa v ňom opakujú definície elementu `<event>`, ale to opravíme neskôr.

```xml
<?xml version="1.0"?>

<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:example:calendar" elementFormDefault="qualified">
    <element name="event"> ①
        <complexType>
            <sequence>
                <element name="date" type="dateTime" />
                <element name="description" type="string" />
                <element name="participants" minOccurs="0">
                    <complexType>
                        <sequence>
                            <element name="participant" type="string"
maxOccurs="unbounded" />
                        </sequence>
                    </complexType>
                </element>
            </sequence>
        </complexType>
    </element>

    <element name="calendar">
        <complexType>
            <sequence>
                <element name="event" minOccurs="0" maxOccurs="unbounded"> ①
                    <complexType>
                        <sequence>
                            <element name="date" type="dateTime" />
                            <element name="description" type="string" />
                            <element name="participants" minOccurs="0">
                                <complexType>
                                    <sequence>
                                        <element name="participant" type="string"
maxOccurs="unbounded" />
                                    </sequence>
                                </complexType>
                            </element>
                        </sequence>
                    </complexType>
                </element>
            </sequence>
        </complexType>
    </element>
</schema>
```

① Definícia elementu sa opakuje, pretože je naozaj rovnaká v samostatnom dokumente i v zozname udalostí v kalendári.

Schéma s opakujúcimi sa elementami

Element `<event>` v predošlom príklade sa vyskytuje na dvoch rozličných miestach: buď ako koreňový element alebo ako súčasť kalendára.

Ak chceme zrecyklovať, či znovupoužiť definíciu bez jej opakovania, vytiahnime definíciu štruktúry tohto elementu von, mimo elementov, a následne sa na ňu odkážeme z oboch miest.

Podobne ako v bežnom programovaní tried, či štruktúr `struct` môžeme definovať štruktúru elementu ako samostatný pomenovaný typ.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar"
        elementFormDefault="qualified"
        xmlns:cal="urn:example:calendar"> ③

    <complexType name="Event">  ①
        <sequence>
            <element name="date" type="dateTime" />
            <element name="description" type="string" />
            <element name="participants" minOccurs="0">
                <complexType>
                    <sequence>
                        <element name="participant" type="string"
maxOccurs="unbounded" />
                    </sequence>
                </complexType>
            </element>
        </sequence>
    </complexType>

    <element name="calendar">
        <complexType>
            <sequence>
                ②
                <element name="event" type="cal:Event"
                         minOccurs="0"
                         maxOccurs="unbounded" />
            </sequence>
        </complexType>
    </element>

    <element name="event" type="cal:Event" />  ④
</schema>
```

① Deklaráciu sme presunuli do elementu `<complexType>`, ktorý sme pomenovali `Event` a dali sme mu rovnakú štruktúru udalosti kalendára, ako v predošlých príkladoch.

Dôležitá je jedna vec: typ `Event` patrí do menného priestoru `urn:example:calendar`, čo je určené atribútom `targetNamespace`.

② Element vo vnútri kalendára, teda `<event>` už neuvádza svoju vnútornú štruktúru explicitne, ale odkazom na komplexný typ.

Takýto odkaz však musíme urobiť nepriamo, okľukou cez *prefix menného priestoru*. Spomenuli sme, že typ `Event` patrí do menného priestoru `urn:example:calendar` (jeho kvalifikované meno je `{urn:example:calendar}Event`). Keďže menné priestory môžu byť mimoriadne dlhé — napríklad `http://www.w3.org/2001/XMLSchema` — musíme použiť ich zástupné mená (aliasy), teda **prefixy**. V atribúte `type` sme sa rozhodli použiť prefix `cal:`. Musíme však ešte určiť, že `cal:` je prefix pre `urn:example:calendar`.

③ Mapovanie medzi menným priestorom `urn:example:calendar` a jeho prefixom `cal` urobíme v koreňovom elemente pomocou klasického mechanizmu menných priestorov. Prefix menného priestoru je ľubovoľný, my sme sa rozhodli pre krátky a úderný `cal`.

④ Voľne stojaci element `<event>` ako koreňový element je tiež typu `cal:Event`. Platí podobná filozofia: použijeme odkaz na komplexný typ `Event`, pričom jeho menný priestor je určený prefixom.

Takýto štýl schémy sa nazýva **žalúzia** (*Venetian Blind*). Koreňové elementy sú globálne, všetky ostatné elementy sú lokálne. Viacnásobne používané štruktúry sú deklarované cez pomenované komplexné či jednoduché typy.

Schéma s vlastnými jednoduchými typmi

Jednoduché typy podporujú rozličnú sadu špeciálnych obmedzení. Reťazce s dĺžkou v danom rozsahu, čísla v danom intervale, iné reťazce spĺňajúce formát v tvare regulárneho výrazu, či hodnoty z daných možností.

XML Schema umožňuje definovať typy pomocou reštrikcií (*restrictions*) a faziet (*facets*).

Predstavme si, že chceme obmedziť popis udalosti na 32 znakov. V schéme dodáme vlastný jednoduchý dátový typ. Vložíme ho priamo pod element `<schema>`:

```
<simpleType name="Description">
    <restriction base="string">
        <maxLength value="32" />
    </restriction>
</simpleType>
```

Jednoduchý typ (*simple type*), ktorý určuje formát hodnôt v elemente, vznikol:

- ako reštrikcia zabudovaného dátového typu `string` (reťazec). Reštrikcie deklarujeme v elemente `<restriction>`
- s jednou fazetou, ktorá obmedzí dĺžku na 32 znakov. Tá je uvedené v elemente `<maxLength>`.

Následne vieme upraviť dátový typ v elemente `<description>`. Namiesto reťazcového typu použijeme odkaz na typ `Description`:

```
<element name="description" type="cal:Description" />
```

Podobne ako v prípade zložených dátových typov použijeme plne kvalifikovaný odkaz, kde menný priestor uvedieme pomocou prefixu `cal`.

Obmedzenie začne platiť pre ľubovoľný element `<description>`, bez ohľadu na to, či je v samostatnom dokumente `<event>` alebo v rámci udalosti kalendára.

Nasledovný dokument prestane zodpovedať schéme:

```
<event xmlns="urn:example:calendar">
    <date>2019-05-30T10:00:00</date>
    <description>A Very Long Conference Name With More Than 32
Characters</description>
</event>
```

Pri pokuse o validáciu uvidíme chyby indikujúce porušenie facetov a reštrikcí:

```
Element '{urn:example:calendar}description': [facet 'maxLength'] The value has a
length of '56'; this exceeds the allowed maximum length of '32'.
Element '{urn:example:calendar}description': 'A Very Long Conference Name With More
Than 32 Characters' is not a valid value of the atomic type
'{urn:example:calendar}Description'.
```

Inštancie a automatické priradenie schémy

Každá inštancia dokumentu môže mať implicitne priradenú schému, oproti ktorej sa dá zvalidovať.

```
---
<calendar xmlns="urn:example:calendar"
          xsi:schemaLocation="urn:example:calendar calendar.xsd"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
```

Koreňovému elementu môžeme priradiť atribút `schemaLocation`. Jeho hodnota pozostáva z dvojíc oddelených medzerami, napr. `urn:example:calendar calendar.xsd`

- prvá časť dvojice reprezentuje **menný priestor**, ku ktorému priradíme XML schému
- druhá časť dvojice predstavuje adresu **URL**, na ktorej sa nachádza XML schéma k predošlému mennému priestoru. Adresa môže byť absolútna, napríklad `https://www.w3.org/2009/XMLSchema/XMLSchema.xsd` alebo relatívna vzhľadom k inštancii XML dokumentu. V príklade máme relatívnu adresu, kde sa očakáva, že schéma je v rovnakom "adresári" ako samotná inštancia. Validátor pristúpi, či stiahne schému dostupnú na danej adrese a použije ju pri validácii.

Samotný atribút `schemaLocation` je plne kvalifikovaný a patrí do menného priestoru `http://www.w3.org/2001/XMLSchema-instance`. Ak ho chceme použiť v dokumente, musíme sa naňho odkázať cez prefix menného priestoru, ktorý je podľa konvencie `xsi`. To je dôvod, prečo musíme deklarovať mapovanie medzi prefixom a menným priestorom:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

Alternatívne zápisy schémy

Alternatívny obvyklý zápis XML schémy využíva explicitný prefix menného priestoru:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:cal="urn:example:calendar"
           targetNamespace="urn:example:calendar"
           elementFormDefault="qualified">
    <!-- ... -->

    <xs:simpleType name="Description">
        <xs:restriction base="xs:string">
            <xs:maxLength value="32" />
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

Doposiaľ sme mali zavedený implicitný menný priestor `http://www.w3.org/2001/XMLSchema`, čo znamenalo, že všetky elementy z jazyka **XML Schema** sme mohli uviesť bez prefixu menného priestoru. Mnoho XML schém však používa explicitný prefix, ktorým je obvykle `xs`, či `xsd`.

V takom prípade musíme:

1. Zaviesť mapovanie prefixu na menný priestor. V koreňovom elemente uvedieme:

   ```
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   ```

2. Všetky elementy patriace do menného priestoru jazyka *XML Schema* musia byť uvedené s prefixom: napríklad `<xs:schema>`, či `<xs:complexType>`.
3. Všetky dátové typy z jazyka *XML Schema* musia byť uvedené s prefixom, napríklad `xs:string`, či `xs:dateTime`.

V ukážke vidíme, ako sa element reštrikcie `<xs:restriction>` odvodil od zabudovaného elementu reťazec (*string*), na ktorý sa odkážeme pomocou prefixu, teda `xs:string`.

XML schéma, kde konštrukčné elementy (`element`, `complexType` atď.) sú v mennom priestore s prefixom `xs`, resp. `xsd`, je ekvivalentná schéme, kde sú konštrukčné elementy v implicitnom mennom priestore. Jediný rozdiel uvidíme v prípade, že chceme konštruovať schému pre dokumenty, u ktorých elementy nepatria do žiadneho menného priestoru (atribút `targetNamespace` vynecháme). To je veľmi okrajová situácia, ktorá sa neodporúča použiť a ak áno, konštrukčné elementy musia mať explicitný prefix.

Schéma s elementami, ktoré majú atribúty

Komplexné elementy s atribútmi

Elementy popisované schémou môžu mať svoje vlastné atribúty. Tie často popisujú *metadáta*, teda dáta o dátach reprezentovaných v dokumente.

Zoberme si dokument, kde atribút `app` hovorí o aplikácii, ktorá vytvorila príslušný kalendár.

```
<calendar xmlns="urn:example:calendar" app="TurboCalendar">
    ...
</calendar>
```

Atribút `app` je reťazcový. Poďme ho teraz zareprezentovať v schéme.

Element s atribútami musí byť vždy komplexný (`complexType`). Ak chceme jednoduchý typ (*simple type*) s atribútom, musíme ho deklarovať ako komplexný typ.

```
<element name="calendar">
    <complexType>
        <sequence>
            <element name="event" type="cal:Event" minOccurs="0" maxOccurs="unbounded"
/>
        </sequence>
        <attribute name="app" type="string" use="required"/> ①
    </complexType>
</element>
```

① V elemente `<calendar>` deklarujeme atribút `app` typu reťazec (*string*).

Atribút `use` určuje povinnosť atribútu. Hodnota `required` vraví, že atribút je povinný. (Ďalšie možnosti sú: implicitný `optional` pre nepovinné atribúty a `prohibited` pre zakázaný atribút.)

Napriek tomu, že v XML dokumente sú atribúty uvedené pred vnorenými elementami, v XML schéme najprv uvádzame podelementy (v príklade `<sequence>`) a až následne uvádzame atribúty.

Jednoduché elementy s atribútmi — rozšírenia typov pomocou *extensions*

Ak chceme ukázať jednoduché elementy (bez vnorených elementov) s atribútmi, musíme sa vrátiť k úplne prvému príkladu. Dodajme doňho atribút `app`.

```
<calendar xmlns="urn:example:calendar" app="iCal">3 events</calendar>
```

Schéma, ktorá popisuje uvedený jednoduchý dokument, musí zadeklarovať `<calendar>` ako komplexný typ, a to i napriek tomu, že obsah je jednoduchý.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:example:calendar">
    <element name="calendar">
        <complexType>
            <simpleContent> ①
                <extension base="string"> ②
                    <attribute name="app" type="string" use="required"/> ③
                </extension>
            </simpleContent>
        </complexType>
    </element>
</schema>
```

① Element zadeklarujeme ako komplexný typ, ale s jednoduchým obsahom (*simple content*), ktorý zakazuje podelementy.

② Element následne použijeme ako rozšírenie (**extension**) existujúceho jednoduchého typu — v našom prípade reťazca — ktorému dodáme ďalšie atribúty.

③ Elementu `<calendar>` dodáme povinný (*required*) reťazcový (*string*) atribút s názvom `app`.

Rozšírenie (**extension**) plní v XML schéme podobnú funkciu ako dedičnosť v objektovo orientovanom programovaní. Typ, ktorý rozširujeme ("od ktorého dedíme") predstavuje akúsi šablónu, ktorú obohatíme o nové atribúty, či podelementy.

Jednoduché typy s atribútmi a facetmi: použitie reštrikcie a extenzie pre jeden element

Niekedy chceme, aby element obsahoval naraz aj atribúty, ale podliehal špeciálnym obmedzeniam (napríklad na dĺžku, či formát).

V taktom prípade použijeme trik:

1. Vytvoríme vlastný dátový typ s reštrikciou.
2. Použijeme ho ako dátový typ elementu, ktorý rozšírime o atribúty.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar"
        xmlns:cal="urn:example:calendar">

    <simpleType name="CalendarSummary"> ①
        <restriction base="string"> ②
            <pattern value="\d+ event(s)?" /> ③
        </restriction>
    </simpleType>

    <element name="calendarSummary">
        <complexType>
            <simpleContent> ③
                <extension base="cal:CalendarSummary"> ④
                    <attribute name="app" type="string" use="required"/>
                </extension>
            </simpleContent>
        </complexType>
    </element>
</schema>
```

① Deklarujeme vlastný dátový typ `CalendarSummary` reprezentujúci element s jednoduchým textovým obsahom.

② Obsah elementu vytvoríme reštrikciou typu reťazec

③ Fazetou reštrikcie bude regulárny výraz.

④ Sumár kalendára (koreňový element) definujeme ako komplexný typ s jednoduchým obsahom, ktorý založíme na dátovom type `CalendarSummary`. Extenziou nášho vlastného dátového typu získame element s formátom vyhovujúcim regulárnemu výrazu, ktorému vieme pridať dodatočné atribúty.

Inštancia dokumentu XML, ktorý vyhovuje schéme:

```
<calendarSummary xmlns="urn:example:calendar" app="iCal">3 events</calendarSummary>
```

Takýto dokument s jedným elementom podporuje i atribút `app`, i predpis na obsah zodpovedajúci regulárnemu výrazu.

Kompletná XML schéma

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar"
        elementFormDefault="qualified"
        xmlns:cal="urn:example:calendar"
>
    <element name="event" type="cal:Event" />

    <element name="calendar">
        <complexType>
            <sequence>
                <element name="event" type="cal:Event" minOccurs="0"
maxOccurs="unbounded" />
            </sequence>
            <attribute name="app" type="string" use="required"/>
        </complexType>
    </element>

    <complexType name="Event">
        <sequence>
            <element name="date" type="dateTime" />
            <element name="description" type="cal:Description" />
            <element name="participants" minOccurs="0">
                <complexType>
                    <sequence>
                        <element name="participant" type="string"
maxOccurs="unbounded" />
                    </sequence>
                </complexType>
            </element>
        </sequence>
    </complexType>

    <simpleType name="Description">
        <restriction base="string">
            <maxLength value="32" />
        </restriction>
    </simpleType>
</schema>
```

Práca s externými schémami

XML Schema podporuje dva spôsoby práce s externými schémami:

- **include**, kde vieme do existujúcej schémy "vložiť" definície z inej schémy, ale rovnakého menného priestoru tak, ako keby boli v nej uvedené priamo.
- **import**, kde vieme do existujúcej schémy dotiahnuť definície z inej schémy a iného menného priestoru.

Inklúzia externých schém

Inklúzia schémy "skopíruje" obsah externej schémy do aktuálnej schémy. Inklúdovať môžeme len elementy z rovnakého menného priestoru ako má cieľový menný priestor (*target namespace*) aktuálnej schémy.

Predstavme si základnú schému pre udalosti:

event.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar"
        elementFormDefault="qualified"
        xmlns:cal="urn:example:calendar">
    <complexType name="Event">
        <sequence>
            <element name="date" type="dateTime" />
            <element name="description" type="string" />
        </sequence>
    </complexType>
    <element name="event" type="cal:Event" />
</schema>
```

Schéma má cieľový menný priestor `urn:example:calendar` a deklaruje v ňom jediný globálny element `<event>`.

A teraz si vytvorme druhú schému, `calendar.xsd`, ktorá chce využiť existujúce deklarácie z externej schémy `event.xsd`.

calendar.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar"
        elementFormDefault="qualified"
        xmlns:cal="urn:example:calendar">

    <include schemaLocation="event.xsd" /> ①

    <element name="calendar"> ②
        <complexType>
            <sequence>
                <element name="event" type="cal:Event" maxOccurs="unbounded"/> ③
            </sequence>
        </complexType>
    </element>
</schema>
```

① Schéma `calendar.xsd` má cieľový menný priestor zhodný s cieľovým menným priestorom schémy `event.xsd` (ide o priestor `urn:example:calendar`). Môžeme teda do nej priamo vložiť (*include*) obsah externej schémy. Inklúziu zrealizujeme elementom `<include>` a uvedením

absolútnej adresy URL alebo relatívnej adresy k externej schéme.

② V schéme si deklarujeme vlastný element `<calendar>`, ktorý bude obsahovať zoznam udalostí `<event>`.

③ Štruktúra každého elementu sa riadi komplexným typom `Event` deklarovaným v schéme `event.xsd`

Zo schémy môžeme sa môžeme odkazovať len na globálne typy a elementy.

Referencie na elementy z externej schémy

Elementy pre udalosti môžeme použiť aj iným spôsobom, odkazom. Namiesto deklarácie elementu a odkazu na jeho typ môžeme uviesť **referenciu**:

```
<element ref="cal:event" maxOccurs="unbounded"/> ①
```

① Referenciu na globálny element realizujeme atribútom `ref`, kde uvedieme kvalifikovaný názov elementu použiteľného na príslušnom mieste. (Kvalifikovaný názov uvedieme pomocou prefixu, ktorý musí byť namapovaný na príslušný menný priestor, ideálne v koreňovom elemente schémy.)

Import externých schém

Import schémy vezme elementy a typy z menného priestoru externej schémy a sprístupní ich v aktuálnej schéme.

Importované súčasti musia byť z iného menného priestoru ako má cieľový menný priestor aktuálnej schémy!

Predstavme si základnú schému pre udalosti:

event.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:event"
        xmlns:e="urn:example:event"
        elementFormDefault="qualified"> ①

    <complexType name="Event"> ②
        <sequence>
            <element name="date" type="dateTime" />
            <element name="description" type="string" />
        </sequence>
    </complexType>
    <element name="event" type="e:Event" /> ③
</schema>
```

① Schéma `event.xsd` má cieľový menný priestor `urn:example:event` namapovaný na prefix `e`.

② V schéme deklarujeme komplexný typ `Event` s dvoma lokálnymi podelementami pre dátum a popis.

③ Deklarujeme globálny element `<event>`, ktorého štruktúra sa riadi komplexným typom `Event`. Tento element má kvalifikované meno `{urn:example:event}event`.

Vytvorme teraz druhú schému pre kalendár:

calendar.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified"
        targetNamespace="urn:example:calendar"
        xmlns:e="urn:example:event"> ①

    <import namespace="urn:example:event" schemaLocation="event.xsd" /> ②

    <element name="calendar">
        <complexType>
            <sequence>
                <element ref="e:event" maxOccurs="unbounded"/> ③
            </sequence>
        </complexType>
    </element>
</schema>
```

① Schéma pre kalendár `schema.xsd` deklaruje elementy do cieľového menného priestoru `urn:example:calendar`. Zároveň deklarujeme mapovanie prefixu `e` na menný priestor udalostí z importovanej schémy.

② Schému pre udalosti `event.xsd` zavedieme do aktuálnej schémy. Keďže menný priestor tejto externej schémy je odlišný od cieľového menného priestoru aktuálnej schémy, použijeme `<import>`. Uvedieme adresu schémy (`schemaLocation`) a menný priestor, do ktorého importneme jej prvky.

③ Elementy kalendára znovupoužijeme zo schémy. Použijeme *referenciu* na globálny element `event` zo schémy pre udalosti. Odkaz samozrejme uvedieme v kvalifikovanom tvare, s použitím prefixu `e`.

Pozrime sa teraz na vzhľad inštancií:

calendar.xml

```
<calendar xmlns="urn:example:calendar"
          xmlns:e="urn:example:event"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="urn:example:calendar calendar.xsd">①

    <e:event> ②
        <e:date>2019-05-30T09:00:00</e:date>
        <e:description>Test</e:description>
    </e:event>
</calendar>
```

① Kalendár má koreňový element s celým kvalifikovaným menom `{urn:example:calendar}calendar`. Validácia sa bude riadiť schémou `schema.xsd`.

② Vnorené elementy však patria do "externej schémy" pre udalosti, ktorá má odlišný menný priestor. Keďže podľa schémy `event.xsd` patrí element `<event>` do menného priestoru `urn:example:event`, musíme ho uviesť s korektným prefixom (v našom príklade `e`). Nezabudnime na to, že import zlučuje elementy z dvoch odlišných menných priestorov, čo je dôvod, prečo sa prefixy líšia.

V prípade, že sa v inštancii zídu dva menné priestory, môže pomôcť explicitné uvedenie prefixov. Dokument potom môže vyzerať nasledovne:

calendar.xml

```
<c:calendar xmlns:c="urn:example:calendar"
            xmlns:e="urn:example:event"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="urn:example:calendar calendar.xsd">
    <e:event>
        <e:date>2019-05-30T09:00:00</e:date>
        <e:description>Test</e:description>
    </e:event>
</c:calendar>
```

- Kalendárový element má kvalifikované meno `{urn:example:calendar}calendar` a má prefix `c`.
- Udalostný element má kvalifikované meno `{urn:example:event}event` a má prefix `e`. Oba prefixy sme namapovali na príslušné menné priestory v koreňovom elemente inštancie.

Literatúra a zdroje

- XML Schema Part 0: Primer Second Edition. Jednoduchý úvod do XML schém od autorov špecifikácie.
- What does `elementFormDefault` do in XSD?. Vysvetlenie pravidla o kvalifikovaní elementov.

- Introducing Design Patterns in XML Schemas. Návrhové vzory pri tvorbe XML schém.