Príručka XML

IT ACADEMY

IT Academy s.r.o., Tomášikova 50/A, 831 04 Bratislava, tel.: 0917/095 406, 0907/375 543
IČO: 46 759 786, DIČ: 2023556766, IČ DPH: SK2023556766

Obsah

Túto príručku môžete využiť ako pomôcku pri práci s XML. **Príručka podlieha autorským právam a jej vlastníkom je spoločnosť IT Academy s.r.o.**

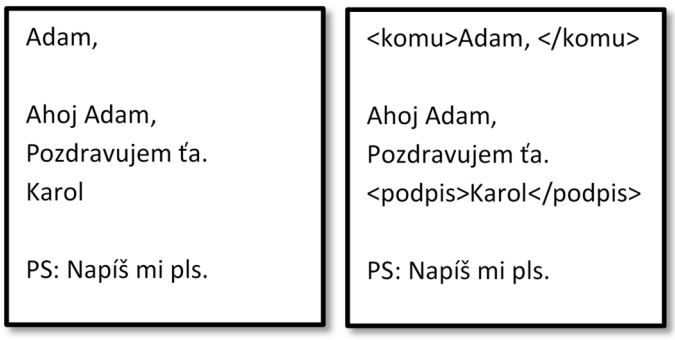
I. Princípy formátu XML

Formát XML definovalo **konzorcium** WC3 ako formát pre prenos všeobecných dokumentov a dát. XML je skratka pre eXtensible Markup Language, t.j. rozšíriteľný **značkovací** jazyk. Návrh XML vychádza zo **staršieho** a všeobecnejšieho štandardu SGML. Zo štandardu SGML vychádzal aj formát dokumentov **HTML**.

Sada značiek formátu **HTML** je pevná a súži k vyjadreniu prezentačnej podoby dokumentov. Naproti tomu v **XML** sada značiek pevná nie je, ale môže byť definovaná pre rôzne sady dokumentov rôzne. Definícia sady značiek môže byť **súčasťou** definície XML dokumentu, môže byť **špecifikovaná** odkazom alebo môže byť **dohodnutá** dopredu.

Značky majú tvar všeobecných zátvoriek ako: <podpis>Jozef</podpis>

Značky slúžia k **označeniu** určitých prvkov dokumentu. Značky majú **otváraciu** zátvorku (start-tag), napr. <podpis> a zatváraciu zátvorku (end-tag), napr. </podpis>. Pokiaľ je text medzi zátvorkami prázdny, je možné dvojicu **otváracích** a zatváracích zátvoriek **nahradiť** prázdnym **elementom**, v našom prípade <podpis/>.



Obr. 1 Príklad neoznačkovanej a označkovanej správy

Pokiaľ vyznačíme v tejto správe zátvorkami podpis **odosielateľa**, bude taký dokument lepšie uchopiteľný a bude ho možné **spracovať** aj programom. V **XML** môžeme naviac stanoviť, že dokument typu „správa“ musí obsahovať adresu, oslovenie, text, podpis a dodatok. Túto požiadavku zapíšeme napr. pomocou nástroja **DTD nasledovne**:

<!ELEMENT správa (adresa, oslovenie, text, podpis, dodatok)>

Vlastný obsah **každej** správy potom musí mať **odpovedajúci** tvar, napr.:

```
<sprava>
   <adresa>Adam Šangala</adresa/>.
   <oslovenie>Ahoj Adam!</oslovenie>
   <text>Pozdravujem ťa.</text>
   <podpis>Karol</podpis>
   <dodatok>Napíš mi pls!</dodatok>
</sprava>
```

Pomocou značiek XML vyznačíme syntaktickú štruktúru dokumentu. Sémantika značiek ani význam ich obsahu nie sú **pomocou** XML definované. Napr. pre vyššie uvedenú správu nevieme, čo obsah správy znamená, **nevieme**, ako správu spracovať – to je vecou **aplikácie**, ktorá s **dokumentom manipuluje**.

Význam XML **spočíva** v to, že štruktúra dokumentu je známa, je možné ju kontrolovať a spracovať všeobecnými nástrojmi. Ľubovoľná **aplikácia** si môže štruktúru dokumentu zistiť a podľa tejto štruktúry ich spracovať – napr. zistiť, kto správu **podpísal**. Potreba nezávislého formátu pre reprezentáciu a **prenos** všeobecných dokumentov je **nesporná**.

II. Formát XML

XML formát je pre reprezentáciu a prenos všeobecných dokumentov. Pri návrhu XML sa autori z konzorcia W3C riadili nasledujúcimi princípmi:

- **formát XML** musí byť **použiteľný** v rámci internetu,
- **formát XML** by mal **podporovať** širokú škálu aplikácií,
- **formát XML** musí byť **kompatibilný** s formátom SGML,
- musí byť **ľahké vytvárať** programy, ktoré **manipulujú** s dokumentmi v XML,
- množstvo **variant XML** by malo byť **minimálne**,
- **XML** dokumenty by mali byť **čitateľné** a **pochopiteľné** aj pre človeka.

Na základe týchto princípov navrhli **definíciu** XML, ktorá zahŕňa dve **časti**:

- **definíciu**, čo je to **XML** dokument,
- **definíciu programov**, ktoré spracovávajú XML dokumenty – XML procesorov.

III. XML dokument

XML **dokument** je určitým spôsobom usporiadaná postupnosť znakov istej abecedy. Implicitne sa predpokladá **Unicode** – kód ISO/IEC 10646 [3]. Na rozdiel od **formátov** HTML, rozlišuje XML **dôsledne** malé aj veľké písmená. Nasledujúce konštrukcie preto nie je dobre vytvorený XML dokument, pretože otváracia a **zatváracia** zátvorka sa líši:

```
<Podpis> … </podpis>
```

Pripomeňme si, že **procesory** HTML pripúšťajú obvykle voľnejšiu syntax. Koncové zátvorky je možné vynechať, v značkách sa **nerozlišujú** malé a veľké písmená, prehliadač obvykle zobrazí aj nie celkom korektný **dokument**, napr.:

```
<HTML>
  <body>Pozdrav ťa!</Body>
</html>
```

Značky XML môžu využívať aj národnú abecedu, všeobecne však tento spôsob značenia sa neodporúča. **Nasledujúca** ukážka ilustruje použitie **lokálnej** abecedy v **značkách**.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<správa>
  <podpis>Karol</podpis>
</správa>
```

Fyzicky sa XML **dokument** skladá z postupnosti prvkov nazývaných entity. Fyzická entita ešte nie je logický element **dokumentu**. Z hľadiska procesorov **XML** môže každá fyzická entita obsahovať buď rozpoznateľné dáta alebo **nerozpoznateľné**.

Nerozpoznateľné dáta môžu byť textové alebo binárne, ktoré sa buď procesoru nepodarí interpretovať ako znaky alebo sa jedná o dáta pre inú **aplikáciu**. Rozpoznateľné dáta sú zostavené zo znakov zvolenej abecedy a predstavujú buď znakové dáta alebo **značky**. Značky vyznačujú logickú štruktúru dokumentu a tým aj jeho **zloženie**.

Dokument začína entitou **nazývanou** koreň. Logicky sa XML dokument skladá z prológu, deklarácií, elementov, **komentárov** a inštrukcií pre spracovanie inými aplikáciami. Logické elementy sú v dokumente vyznačené značkami. Sada **značiek** je všeobecne **ľubovoľná**, môže byť ale predpísaná **deklaráciami**.

Deklarácie nie sú povinné, pokiaľ ale chceme **štruktúru** dokumentu kontrolovať, musíme ju deklaráciami predpísať. XML **dokument** je dobre vytvorený, pokiaľ všetky rozpoznateľné entity v dokumente sú správne vytvorené a naviac, všetky **rozpoznateľné** entity, na ktoré existujú v dokumente odkazy, sú rovnako dobre **vytvorené**.

Každá dvojica zátvoriek musí byť **korektne** spárovaná v rámci elementu a tieto dvojice musia byť dobre vnorené do seba – zátvorky sa nesmú krížiť. Z toho plynie, že dobre **vytvorený** XML dokument má **stromovú** štruktúru.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<fax>
  <odosielatel>
    <meno>Adam Šangala</meno>
    <cislo>123 333 321</cislo>
  </odosielatel>
  <adresat>
    <meno>Laco Novomestský</meno>
    <cislo>666 123 666</cislo>
  </adresat>
```

XML **poskytuje** možnosť definície použiteľnej sady značiek, definíciu logickej štruktúry a rozloženie dokumentu – XML schéma **dokumentu**. Jednoduchým nástrojom na definíciu je definícia typu dokumentu DTD. Každý dobre **vytvorený** XML dokument potom môže byť validný, pokiaľ spĺňa obmedzenie správnosti dané **špecifikáciou** XML **schémy**.

Validita dokumentu preto **vyžaduje** overenie proti definovanej špecifikácií obsahu. Rozpoznateľné entity XML **dokumentu** sú zostavované zo znakov. Implicitnou znakovou sadou je Unicode, každý XML procesor preto musí byť schopný akceptovať na výstupe Unicode zakódovaný **pomocou** kódovania UTF-8, **prípadne** UTF-16LE, či UTF-16BE.

Rozlíšenie medzi týmito **dvoma** kódmi by malo byť možné tak, že dokument v UTF-16 musí začínať #XFEFF. Pre **komunikáciu** sa však často používa UTF-8, ktorý je kompatibilný s kódom ASCII v tom zmysle, že obsahuje všetky znaky všetkých **abecied** ASCII a ďalšie znaky sú kódované **pomocou** 2-4 bitov.

XML procesor je **modul**, ktorý vie čítať XML dokumenty a **sprístupňuje** ich prvky aplikáciami. XML procesor kontroluje, či je dokument dobre vytvorený. **Prerušenie** týchto pravidiel predstavuje fatálnu chybu, ktorú musí XML procesor detegovať a **hlásiť** aplikácií.

Pre detekciu **fatálnych** chýb môže XML procesor **poskytovať** ďalšia dáta aplikácií, ale nesmie pokračovať v normálnom spracovávaní. XML **procesor** môže byť validujúci – potom môže kontrolovať, či je **vstupný** XML dokument validný vzhľadom k zadanej špecifikácií štruktúry. Validujúci XML procesor musí hlásiť nezhody s **deklarovanou** štruktúrou **XML** dokumentu.

IV. Elementy a atribúty

XML dokument **obsahuje** jeden alebo viac elementov, vymedzených zátvorkami. Každý element je identifikovaný menom, ktoré je uvedené v **zátvorkách** obmedzujúcich element, napr. element osoba. Pokiaľ element nie je prázdny, tvorí jeho **obsah** entity uzavreté medzi **otváracou** a **uzatváracou** zátvorkou.

```
<osoba>
   Text tvoriaci obsah elementu typu osoba.
</osoba>
```

Obsahom **elementov** môže byť ľubovoľná **postupnosť** znakových dát, iných vnorených elementov, referencií na iné odkazy, sekcií CDATA, **inštrukcií** pre spracovanie inou aplikáciou alebo **komentárov**.

Elementy môžu mať **atribúty**, ktorými sú bližšie **charakterizované**. Atribúty sa vkladajú do otváracej **zátvorky** elementu vo forme parametrov zadaných pomocou kľúčových slov. Každá špecifikácia atribútu má meno a **hodnotu** – zapisuje sa ako dvojica meno=”hodnota”. Hodnota atribútu musí byť vždy uvedená v **úvodzovkách** alebo **apostrofoch**.

Vždy sa jedná o **reťazec** znakov, ktoré je možné interpretovať napr. ako celé číslo. Už zmienenú faxovú správu môžeme **pomocou** atribútov **vyjadriť** napr. nasledovne:

```
<fax>
  <odosielatel meno="Adam Šangala"
    cislo="123 333 321"> … </odosielatel>
  <adresat meno="Laco Novomestský"
    cislo="666 123 666"> … </adresat>
  <text> … </text>
</fax>
```

V. Prázdny element

Prázdne elementy **nemajú** žiadny obsah. Slúžia pre **vyznačenie** miesta v dokumente, napr. vyznačené miesta, kam sa neskôr doplní nejaký prvok. Prázdny **element** môže mať atribúty, ktoré **popisujú** jeho vlastnosti – napr. to môže byť odkaz na objekt, ktorý sa má do daného miesta vložiť.

Rôzne formy zápisu **prázdneho** elementu ukazujú nasledujúce príklady, kde prvý element označuje vložený obrázok, zatiaľ čo druhý a tretí sú **alternatívne** zápisy prázdneho elementu vyznačujúce miesto, kam sa vložiť **oddeľovacia čiara**.

```
<IMG align=”left”
src=”http://www.w3.org/Icons/w3c_home”/>
<br></br>
<br/>
```

VI. Komentáre

Komentáre sa v XML **dokumente** môžu vyskytovať kdekoľvek, ale nemôžu byť umiestnené v značkách. Majú svoje **špecifické** zátvorky "<!—" a "-->", aby sa dali odlíšiť od vlastností obsahu dokumentu. Komentáre sa môžu vnoriť. V komentári sa nesmie **vyskytovať** reťazec "—", ktorý sa používa v **komentárových zátvorkách** podľa **syntaxe**:

```
<!-- Ja som príklad komentára -->
```

XML procesor môže **umožniť** aplikácií **čítať** text komentárov. Značky uvedené v tele **komentára** sa však interpretujú ako komentár, nie ako označenie elementu. V nasledujúcom komentári sa preto text <príklad> **neinterpretuje** ako otváraciu zátvorku **elementu** príklad, ale ako súčasť textu **komentára**.

```
<!-- Tu bude uvedená definícia elementu <príklad> -
->
```

VII. Inštrukcie pre spracovanie

Definície XML **dokumentu** umožňuje, aby dokument obsahoval inštrukcie, ktoré majú byť spracováva inú **aplikáciu**, než je XML procesor. Takej časti sa hovorí sekcia inštrukcií a používajú sa pre ňu špeciálne zátvorky so znakom „?“. **Inštrukcia** obsahuje návesti, ktorými je **identifikovaná** aplikácia, pre ktorú je **určená**.

Z **pochopiteľných** dôvodov to nemôže byť žiadny variant reťazca “XML”. Za návesťou môže byť biely znak a potom **nasleduje** text, ktorý predstavuje dáta pre danú aplikácií. **Nasledujúca** inštrukcia je napr. **určená** pre aplikáciu xml:

```
<?xml data pre inu aplikáciu?>
```

Obsah sekcie **inštrukcií** nie je chápaný ako súčasť XML dokumentu, je iba predaný ku spracovaniu danej aplikácie. Napr. **pripojenie** štýlu k dokumente **dosiahnem** inštrukcií:

```
<?xml-stylesheet href=”styl.css” type=”text/css”?>
```

VIII. Sekcia CDATA

Niekedy **potrebujeme** do dokumentu vložiť blok textu, ktorý by mohol byť považovaný za značku. Pokiaľ tomu chceme **zabrániť**, použijeme špeciálnu sekciu typu znakové dáta . CDATA, ktorá slúži k značeniu takých miest. Pokiaľ **uvedieme** v dokumente **text**:

```
<pozdrav>Ahoj!</pozdrav>
```

bude chápaný ako element **pozdrav** s obsahom „Ahoj!“. Pokiaľ chceme, aby pozdrav v zátvorkách nebol **chápaný** ako značka, ale aby bol ako text súčasťou obsahu elementu, musíme použiť **konštrukciu**:

```
&lt;pozdrav&gt;Ahoj!&lt;/pozdrav&gt;
```

Pomocou sekcie CDATA to zapíšeme ľahšie ako:

```
<![CDATA[ <pozdrav>Ahoj!</pozdrav>]]>
```

Všeobecne má sekcia CDATA tvar:

```
<![CDATA[text]]>
```

Reťazec „]]“ sa v tele sekcie CDATA pochopiteľne nesmie vyskytovať.

IX. Definícia štruktúry dokumentu pomocou DTD

Každý dobre **vytvorený** CML dokument musí začať prológom, za ktorým nasleduje koreňový element dokumentu. Prológom sa **stanoví** verzia XML, kódovanie dokumentu a prípadne požiadavky na štruktúru **dokumentu**. Každý XML dokument **obsahuje** jeden či viac **elementov**.

Vždy však **obsahuje** práve jeden koreňový element, ktorého žiadna časť nie je obsiahnutá v žiadnom inom **elemente**. Súčasťou dokumentu môže byť aj skôr **zmienená** sada komentárov a inštrukcií pre iné **aplikácie**. Prológ XML dokumentu začína deklaráciou verzie XML. Deklarácia verzie dokumentu má tvar, ktorý pripomína **inštrukciu** pre aplikáciu xml:

```
<?xml version=”1.0”?>
```

Kľúčové slovo **version** umožňuj určiť **použitú** verziu špecifikácie formátu **XML**, nie je to ale atribút elementu – deklaráciou nie je element. V deklarácií je možné ďalej definovať kódovanie dokumentu. Pokiaľ nie je **explicitne** definované, uvažuje sa implicitne **Unicode**, kódovanie UTF-8 alebo UTF-16.

```
<?xml version=“1.0“ encoding=“iso-8859-2”
standalone=”no”?>
```

Ďalšou súčasťou **prológu** môže byť deklarácie typu XML dokumentu, ktorou sa stanoví gramatika pripustenej triedy **dokumentov**. Tejto gramatike sa hovorí definícia typu dokumentu alebo tiež schéma dokumentu. Konkrétne schéma **dokumentu** sa určí rôzne podľa použitého nástroja, v **ktorom** je vyjadrené.

Deklarácia typu môže byť uvedená **priamo** v **XML** dokumente, alebo môže **odkazovať** na externú entitu, ktorá **obsahuje** deklaráciu značiek alebo oboje. Skutočná gramatika dokumentu je tvorená spojením všetkých týchto deklarácií, kde prvá **deklarácia** má prednosť pred **ďalšími**.

Opakovaná **deklarácia** sa nepovažuje za chyby, ignoruje sa, XML procesor však môže pre túto situáciu použiť varovanie. Interné **deklarácie** sa preto uvádzajú ako prvé, aby mali v prípade kolízie prednosť pred **extrémnymi**.

Dokumentu môže byť označený ako **samostatný**. Pokiaľ je takto **deklarovaný**, nesmie obsahovať odkazy na **externé** entity alebo parametre. – musí byť spracovateľný sám o sebe. Každý nesamostatný XML dokument je možné previesť na **samostatný** dokument vložením externých entít priamo do **dokumentu**.

Na rozdiel od **formátu** SGML, kde je schéma dokumentu povinná, v XML dokumente to tak nie je. To umožňuje vytvárať XML dáta tak, že ich **značky** nemusia byť v dobe návrhu aplikácií známy. Inými slovami, môže **navrhovať** aplikácie bez **znalostí** schéma z dát.

X. Príklady definície typu dokumentu

Dokument s lokálnym DTD by mohol vyzerať takto:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<!DOCTYPE pozdrav [
  <!ELEMENT pozdrav (#PCDATA)>
]>
<pozdrav>Ahoj!</pozdrav>
```

zatiaľ čo dokument s externým DTD uloženým v súbore ahoj.dtd napr. takto:

```
<?xml version=“1.0“?>

<!DOCTYPE pozdrav SYSTEM “ahoj.dtd”>
<pozdrav>Ahoj!</pozdrav>
```

Existuje **verziu** formátu HTML označovaná **XHTML**, ktorá predstavuje presne vyjadrenie HTML pomocou **XML**. Dokument v XHTML s definíciou **stanovenou** verejným externým DTD potom môže vyzerať napr. **nasledovne**:

```
<!DOCYTYPE html PUBLIC “///W3C//DTD XHTML 1.0
Transitional//EN”
    “http://www.w3.org/TR/2002/REC/xhtml1/20020801/DTD
    /xhtml-transitional.dtd”>
<html xmlns=“http://www.w3.org/1999/xhtml“>
   <head>
<meta http-equiv=“content-type” content=”text/html”;
charset=”ISO/8859/2”/>
<title>Správ</title>
</head>
<body>Moja správa</body>
</html>
```

Naproti tomu dokument s **definíciou** stanovenou XML schémou v **jazyku** XML **Schéma** môže vyzerať, napr. **takto**:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<pozdrav
xmlns:xsd=”http://www.w3.org/2001/XMLSchema/instanc
e”
      xsd:noNamespaceSchemaLocation=”http://priklad
    .com/pozdrav.xsd”>
Ahoj!</pozdrav>
```

Obsah DTD

v definícií typu **dokumentu** zapísaného pomocou DTD sa môžu vyskytovať:

- **deklarácie** typu **elementu**,
- **deklarácie** zoznamu **atribútov**,
- **deklarácie entity**,
- **deklarácie notácie**.

XI. Deklarácia typu elementu

Deklarácia typu **elementu** zavádza meno elementu a špecifikuje jeho možný **obsah**. Element môže byť **prázdny** (Empty):

```
<ELEMENT BR EMPTY>
```

Element môže mať **ľubovoľný obsah** (ANY):

```
<ELEMENT čokoľvek ANY:
```

Element môže **obsahovať znakové** dáta:

```
<ELEMENT meno (#PCDATA)>
```

Takto je **možné** popísať základné elementy bez ďalších vnútorných štruktúr – t.j. elementy, ktorých obsahom môže byť **reťazec** znakov. Pokiaľ chceme popísať element, ktorého vnútorná štruktúra je zložitejšia, môžeme požiadavky na obsah **elementu** vyjadriť **regulárnym** výrazom.

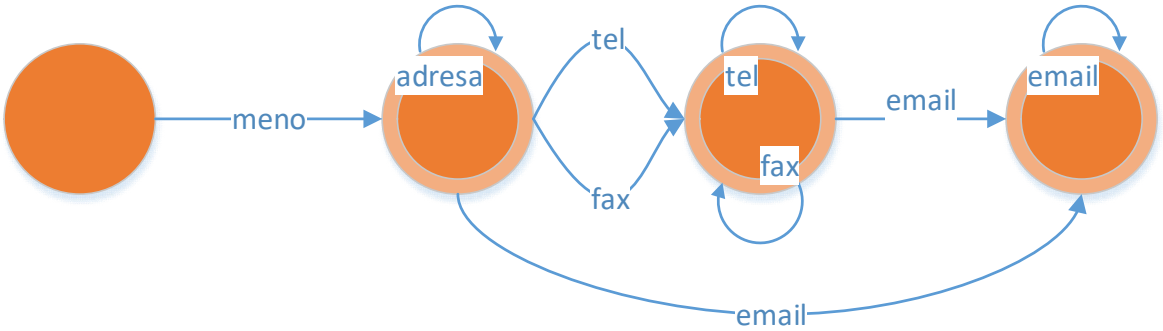
Ním **popíšeme**, z akých zložiek sa obsah elementu môže skladať a ako môžu byť tieto zložky usporiadané. Pre regulárny výraz DTD sa používa **nasledujúca notácia**:

- pomocou **‘,’** sa vyjadruje **sekvencia zložiek**,
- pomocou **‘|’** možnosť **výberu** z **alternatív**,
- pomocou **‘?’** sa vyjadruje **nepovinnosť zložky**,
- pomocou **‘+’** sa **vyjadruje možnosť** opakovania zložky raz a prípadne viackrát,
- pomocou **‘*’** sa vyjadruje **možnosť ľubovoľného** opakovania zložky, vrátane žiadneho.

Operátory '*', '+' a '?' sa niekedy nazývajú **operátory** početnosti. K regulárnym výrazom existuje konečný automat, čo **zjednodušuje** problém konštrukcie validujúceho XML **procesoru**.

meno, adresa*, tel|fax)*, email*

Tento výraz predpisuje **postupnosť**, ktorá začína povinným menom, potom nasleduje ľubovoľný počet adries, potom **ľubovoľný** počet telefónov alebo faxov a na záver ľubovoľný počet e-mailov. Odpovedajúci konečný **automat** je uvedený na **obrázku**.



Obr. 2 Príklad konečného automatu

Ak chceme napr. vyjadriť **požiadavku**, že element odstavec môže obsahovať buď znakové dáta alebo element tučné, vyjadríme túto **skutočnosť** nasledujúcou deklaráciou typu **elementu**:

```
<!ELEMENT odstavec (#PCDATA | tučné)*>
```

Taký **element** má zmiešaný obsah a môže **obsahovať** kombináciu textu a vnorených elementov, napr.:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<!DOCYTYPE odstavec [
    <!ELEMENT odstavec (#PCDATA|tučné)*>
    <!ELEMENT tučné (#PCDATA)>
]>
<odstavec>
  P.S.:<tučné>Napíšte mi!</tučné> niečo
  <tučné>pekné</tučné>.
</odstavec>
```

Deklarácia modelu **zmiešaného obsahu** musí mať vždy nasledujúcu formu:

```
<ELEMENT MENO (#PCDATA | elt1 | elt2 | …)*>
```

Ak si **spomenieme** na štruktúru správy, ktorá obsahuje aspoň jednu adresu, ďalej povinné oslovenie, text a podpis a je **ukončená** nepovinnou **poznámkou**, vyjadríme ju **deklaráciou**:

```
<!ELEMENT správa (adresa+, oslovenie, text, podpis, poznámka?)>
```

Napr. nasledujúca správa je vzhľadom k tejto špecifikácií validná.

```
<sprava>
   <adresa>Adam</adresa>
   <adresa>Karol</adresa>
   <oslovenie>Ahoj!</oslovenie>
   <text>Pozdravujem ťa!</text>
   <podpis>Karol</podpis>
   <poznamka>Napíšte mi!</poznamka>
</sprava>
```

Špecifikácie **štruktúry** pomocou DTD môže byť tiež rekurzívny, ako dokladá nasledujúci **príklad**:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<DOCYTYPE genealogia [
    <!ELEMENT genealogia (osoba*)>
    <!ELEMENT osoba (meno,
        datum_nar,
        osoba?, <!--matka -->
        osoba?)><!--otec-->
…
]>
```

Všimnime si, že **rekurzia** vedie k cyklickým DTD, ktorých spracovanie je ťažšie. Nevýhodou tohto konkrétneho príkladu naviac je, že **pripúšťa** XML dokumenty popisujúce osoby s jedným rodičom bez rozlíšenia, či ide o **matku** alebo **otca**.

XII. Deklarácia atribútov elementu

Deklarácia atribútu **elementu** zavádza meno elementu a jeho **obsah**. **Atribúty** elementu sú DTD zavedené deklarácie množiny **atribútov** podľa nasledujúcej **gramatiky**:

```
<!ELEMENT x …>
<!ATTLIST x meno_atribútu typ_atribútu volitelnosť
implicitná_hodnota>
```

V deklarácií **atribútu** je nutné určiť typy atribútov. Prípustné typy atribútov sú predovšetkým **jednoduché reťazce** – atribúty typu **CDATA**. O každom atribúte je možné stanoviť, či je **povinný**, či **nepovinný**. Povinné atribúty musia byť pre daný element vždy uvedené.

Pokiaľ nie je pri atribúte **uvedené** ani #**REQUIRED** ani **#IMPLIED**, musí mať atribút stanovenú implicitnú hodnotu. Pokiaľ je **implicitná** hodnota atribútu predznačená kľúčovým slovom #FIXED, nie je možné ďalej hodnotu tohto **atribútu** meniť – funguje ako **konštantný** atribút.

V nasledujúcej **špecifikácií** je stanovené, že **element** formulár má fixný atribút metóda s hodnotou “POST”.

```
<!ELEMENT formulár (hlavička, položka*)>
<!ATTLIST formulár metóda CDATA #FIXED “POST”>
```

Po identifikácií **elementov** je možné použiť špeciálny typ atribútu ID, ktorý môže byť pre daný element iba jeden a jeho **hodnoty** musia byť unikátne v rámci daného **XML** dokumentu. **Voliteľnosť** atribútu typu **ID** musí byť explicitne **stanovená**.

Slúži pre **jednoznačnú** identifikáciu elementov v rámci **dokumentov** a odkazovať sa na neho je možné pomocou atribútu typu **IDREF** alebo **IDREFS**. Hodnota atribútu typu IDREF musí byť zhodná s niektorou **existujúcou** hodnotou typu ID –**odkazovať** sa je možné iba na existujúce **elementy**.

Podobne pre **atribút** typu **IDREFS** musia byť všetky jeho hodnoty odkazom na existujúce elementy dokumentu. Typ **IDREFS** reprezentuje viachodnotový atribút, v ktorom sú jednotlivé hodnoty oddelené **bielymi znakmi**.

```
<!ATTLIST kľúč id    ID  #REQUIRED
meno CDATA #implied
```

Pomocou atribútu typu ID, IDREF a –IDREFS je možné špecifikovať genealogické dáta lepšie napr. nasledovne

```
<?xml version=”1.0” encoding=”UTF-8”?>
<!DOCTYPE GENEALOGIA [
    <!ELEMENT genealogia (osoba*)>
    <!ELEMENT osoba (meno,
    <!ELEMENT meno (#PCDATA)>
    <!ATTLIST osoba id ID #REQUIRED
        matka IDREF #IMPLIED
        otec IDREF #IMPLIED
        deti IDREF #IMPLIED
```

Fragmentom XML dokumentu, ktorý obsahuje zodpovedajúce **genealogické** dáta, môže byť nasledujúci **element**:

```
<genealógia>
  <osoba id=”mária” deti=”jana vít”>
    <meno>Mária Nováková</meno>
  </osoba>
  <osoba id=”jozef” deti=”jana vít”>
    <meno>Jozef Novák</meno>
  </osoba>
  <osoba id=”jana” matka=”mária” otec=”jozef”>
    <meno>Jana Nováková</meno>
  </osoba>
  <osoba id=”vít” matka=”mária” otec=”jozef”>
    <meno>Vít Novák</meno>
  </osoba>
<genealógia>
```

Hodnotou **atribútu** typu ID môže byť reťazec zostavený z písmen a čísel a špeciálnych znakov, ktorý začína písmenom alebo **podtržníkom**. Podobne môžeme vytvárať atribúty typu **NMTOKEN**, resp. **NMTOKENS**, ktoré rovnako reprezentujú reťazce zostavené z rovnakých znakov, ale nemusia byť **unikátne** a môžu začínať **číslicou**.

Ďalším možným **typom** atribútu je odkaz na entitu alebo zoznam odkazov na entity – typ je **ENTITY**, resp. ENTITIES. **Hodnotou** atribútu typu **ENTITY** môže byť iba meno entity zavedenej v DTD. Každý prvok atribútu **ENTITIES** musí spĺňať to isté.

Posledným **možným** typom atribútu sú výpočtové typy. Výpočtové typy sú reprezentované zoznamom možných **hodnôt**:

```
<!ATTLIST zoznamy typ
(obrázky|usporiadanie|poznámky)>
```

a prípadnou implicitnou hodnotou uvedenou s zozname:

```
<!ATTLIST osoba titul (žiadný|ing.|dr.) “žiadny”>
```

Referencie na znak

Niekedy nie je možné v texte dokumentu uviesť potrebný znak priamo, pretože by to odporovalo požadovanej **syntaxe**, alebo znak nie je **zobraziteľný** apod. V takom prípade môžete **použiť** referenciu na znak v jednom z **tvarov**:

&#xQ;

&#D;

kde Q je **hexadecimálny** kód znaku v znakovej sade ISO/IE, zatiaľ čo D je dekadicky vyjadrený kód znaku. Nasledujúci fragment ukazuje možný **príklad použitia**:

```
Pre uloženie dát stlačte klávesu <klávesa>menšia-
ako?/klávesa> (&amp;#x3C;).
```

Deklarácia entity

Entity využíva XML podobne ako **symbolické konštanty**. Deklaráciou entity sa stanoví nahradzujúci text pre entitu. Často sa to využíva pri **špeciálnom** znaku, ktorý nie je súčasťou štandardného kódu. Pre špeciálne znaky využijeme v XML existujúce **preddefinované** entity:

```
<!ENTITY lt “&#38;#60;”> <!– znak<-->
<!ENTITY gt “&#62;”> <!– znak> -->
<!ENTITY amp “&#38;#38;”> <!– znak & -->
<!ENTITY apos “&#39;”> <!– znak ; -->
<!ENTITY quot “&#34;”> <!– znak ” -->
```

pomocou ktorých môžeme napr. napísať:

```
<sprava>Pokiaľ má plat &tl; 1000, …</sprava>
```

pretože text obsahujúci iba znak ‘<’ by bol chybný:

```
<sprava>if plat< 1000 then</sprava>
```

Pokiaľ je v **deklarácií** entity priamo uvedený **nahrádzajúci** text uzavretý zátvorkami v **úvodzovkách** alebo **apostrofoch**, považujeme ju za internú. Interná entita je analyzovaná a jej nahrádzajúci text sa získa tak, že všetky **referencie** na znaky a všetky referencie na interné entity v nej **obsiahnuté** sú nahradené ich **telom**.

Rekurzívne odkazy na **seba** samých nie sú povolené. Pokiaľ entita nie je interná, jedná sa o entitu externú. V deklarácií externej **entity** sa vyznačí, či sa jedná o **systémový**, či **verejný objekt**:

```
<!ENTITY open-hatch
    SYSTEM
“http://www.textuality.com/boilerplate/OpenHatch.xml“>
<!ENTITY open-hatch
    PUBLIC “-//Textuality/TEXT Standard open-hatch boilerplate//EN”
          “http://www.textuality.com/biolerplate/OpenHatch.xml”>
```

Za deklaráciou externej entity môže byť uvedený odkaz na notáciu kľúčovým slovom ndata. Notácia musí byť definovaná, odkaz bude napr. v tvare:

```
<!NOTATION gif PUBLIC “Obrázok vo formáte GIF”>
<!ENTITY hatch-pic
    SYSTEM “../grafix/OpenHatch.gif”
    NDATA gif>
```

Pre externé entity sa **predpokladá**, že budú **nahradené** príslušnou **referenciou** URI.

Použitie entity – referencie na entitu

Entity používame pomocou **referencií**, ktorými odkazujeme na definíciu entít. Referencie na entitu začínajú znakom ‘&’ a končia ‘;’. Medzi týmito **obmedzovačmi** sa uvedie meno **entity**.

```
&apos; <-- Referencia na entitu apos = znak apostrof -->
```

Normalizované hodnoty atribútov

Pri spracovaní **hodnôt** atribútov musí XML procesor hodnoty atribútov normalizovať. Podľa definície prebieha normalizácia hodnôt atribútov pre ich validáciou alebo pred predaním ich hodnoty aplikáciám. **Normalizácia** musí prebiehať tak, aby výsledná hodnota **odpovedala** výstupu nasledujúceho **algoritmu**.

Algoritmus 1: Normalizácia hodnôt atribútov

Vstup: Nenormalizovaný výstupný text

Výstup: Normalizovaný text

1. **Spracujte riadky** – všetky hodnoty riadkov musia byť nahradené znakom #xA / nech už je koniec riadku reprezentovaný akoukoľvek kombináciou znakov.
2. **Nastavte normalizovanú** hodnotu ako prázdny reťazec.
3. Pre každý znak, **referenciu** na entitu alebo **referenciu** na znak v nenormalizovanej hodnote atribútu **vykonajte**:
    a. pre referenciu na znak **pripojte** k **výstupu** referovaný znak,
    b. pre referenciu na entitu **aplikujte rekurzívne** krok **3** na všetok nahradzujúci text entity a výsledok pripojte na výstup,
    c. pre biely znak **pripojte** na **výstup** medzeru,
    d. všetky **ostatné znaky** pripojte na výstup priamo.
4. Pokiaľ sa nejdená o atribút **typu** CDATA, vypustíme všetky úvodné medzery a všetky postupnosti medzier nahradíme **medzerou** jednou.

Všimnime si, že referencie na „**nemedzerový** biely **znak**“ v nenormalizovanej hodnote atribútu je nahradený v normalizovanom výstupe týmto znakom, zatiaľ čo samotný biely znak je **nahradený** jednou **medzerou**. V tomto je rozdiel aj v spracovaní referencie na entitu, ktorej nahradzujúci text obsahuje **biele** znaky.

Tie sú **spracované** a **nahradené** v normalizovanom výstupe medzerou. Všetky atribúty, ktoré nie sú **deklarované**, by mali byť spracované **nevalidujúcim** procesorom, ak by boli typu CDATA. Referencie na entitu, ktorá nebola **deklarovaná** je nutne chápať ako **chybu**.

Špecifikácia atribútu	Normalizované ako NMTOKENS	Normalizované ako znakové dáta (CDATA)
a=” xyz”	x y z	#x20 x y z
a=“&d;&d;A&A; &a;B&da;“	A #x20 B	#x20 #x20 A #x20 #x20 #x20 B #x20 #x20
**a=“

A

B
“** | #xD #xD A #xA #xA B
#xD #xA | #xD #xD a #xA #xA B
#xD #xA |

Tab. 1 Normalizácia hodnôt atribútu

Parametre

Parametre sú **podobné** entitám, ktoré v deklarácií aj použitie parametrov je meno parametrov predznačené znakom '%'. Na rozdiel od entít slúži pre zostavenie **DTD**, nie pre vlastný dokument. Na **parametre** sa odkazujeme pomocou referencií. **Referencie** začínajú znakom '%' a končia znakom ';'. Medzi týmito **obmedzovačmi** sa uvedie meno **parametrov**.

```
<!–deklarácia parametru "ISOLat2":-->

<!ENTITY % ISOLat2

SYSTEM "http://www.xml.com/iso/isolat2-
xml.entities">
<!–odkaz na parameter: -->
%ISOLat2;
```

Parametre je možné použiť pri špecifikácií obsahu elementu:

```
<!ENTITY % meno "pozdrav">
<!ENTITY % obsah "(#PCDATA)">
<!ELEMENT %meno; %obsah;>
```

Výsledkom bude popis elementu pozdrav, ako by bol špecifikovaný deklaráciou

```
<!ELEMENT pozdrav (#PCDATA)>
```

Podmienené sekcie

Pri definícií DTD **niekedy** potrebujeme, aby pre XML dokument pripadalo v úvahu niekoľko variant. Napr. potrebujeme, aby kniha v prípravnej **fázy** mohla obsahovať komentáre, ktoré sa vo finálnej verzií nebudú vyskytovať.

To je možné **zariadiť** pomocou podmienenej sekcie, ktorá pomocou kľúčových slov INCLUDE a IGNORE zahŕňa, či ignoruje časť definície dokumentu.

```
<![INCLUDE[
  <! – text, ktorý bude vložený do DTD -->
]]>
<![IGNORE]
  <! – text, ktorý nebude vložený do DTD -->
]]>
```

Problém s rôznymi variantmi knihy cez **parametre** – uvedená verzia je **pracovná** (draft).

```
<!ENTITY % pracovná “NCLUDE”>
<!ENTITY % finálna “IGNORE“>

<![%pracovná;[
  <!ELEMENT kniha (komentár*, názov, obsah,
dodatky?)>
]]>
<![%finálna;[
  <!ELEMENT kniha (názov, obsah, dodatky?)>
]]>
```

Ďalší príklad je definícia matematickej aplikácie XML – jazyka MathML [50].

```
<!—Pramater MathMLStrict je možné použiť pre
striktnú kontrolu -->
<!ENTITY % MathMLstrict “IGNORE”>

<!—Verzia pokiaľ požadujeme striktnú kontrolu-->
<![%MathMLstrict;[
    <!ENTITY %att-mathvariant
     “mathvariant (normal | bold | italic | bold-
                italic | double-struck |bold-fractur
                | script | bold-script | fraktur
                |sans-serif | bold-sans-serif |
                sans-serif-italic |sans-serif-bold-
                italic | monospace)
               #IMPLIED”>
]]>
<!—Verzia pokiaľ nepožadujeme striktnú kontrolu -->
<!ENTITY %  att-mathvariant ”mathvariant CDATA
#IMPLIED”>

<!—Do tejto entity sa definícia att-fontinfo
premietne-->
<!ENTITY % att-fontingo
   “ …
   %att-mathvariant;
   … ”>
```

```
<!—v atribútoch tohto elementu sa att-fontinfo
použije, tj.
medzi atribútmi tohto elementu bude aj atribút
mathvariant, ktorý má typ CDATA alebo vymenovávací
typ. -->
<!ELEMENT %mi.qname; (…)>
<!ATTLIST %mi.qname; %MATHML.Common.attrib;
%att-fontinfo;>
```

Definícia notácie

Definícia **notácie** umožňuje definovať pre určitú aplikáciu systémovú notáciu. Táto aplikácia sa potom využije odkazom na **notáciu** s kľúčovým slovom **NDATA**.

```
<!NOTATION gif SYSTEM
”http://www.../apps/iviewer.exe”>
<!ENTITY logo SYSTEM “.../images/logo.gif” NDATA
gif>
```

XIII. Menné priestory XML

Pokiaľ **potrebujeme** vo formáte XML reprezentovať určitý druh informácie, vytvoríme si obvykle pre tento účel **vlastnú** sadu značiek – značkovací slovník. **Rozhodneme** sa, ako sa budú volať jednotlivé elementy a aké budú mať **atribúty**. Pokiaľ pri špecifikácií štruktúry XML dokumentov používame jazyk DTD, musia byť mená v rámci tejto **špecifikácie** jedinečné.

V DTD neexistuje preťažovanie mien – rovnaké meno nemôžu mať napríklad dva elementy líšiace sa sadou atribútov. Problém nastane, ak chceme schémy kombinovať. V jednom dokumente potom totiž môže byť použitých viac nezávislých sád značiek, v dôsledku čoho môže dôjsť ku konfliktu názvov elementov. Tento problém riešia menné priestory XML.

Uviažme pre ilustráciu nasledujúci fragment XHTML dokumentu popisujúci titul:

```
<titul>
  <tr>
     <td>Matematická analýzy</td>
     <td>Jozef Novák</td><
     td>122.50</td><
  /tr>
</titul>
```

Podobne môžeme mať v XML záznam o jedinom titule v tvare:

```
<titul>
   <autor titul=”Ing” meno=”Jana Nováková”/>
   <názov>Technická dokumentácia</názov>
   <cena>95.00</cena>
</titul>
```

Pokiaľ by sme sa **snažili** „zmiešať“ tieto fragmenty do jedného dokumentu, došlo by ku konfliktu, pretože obsahujú elementy s rovnaký menom, ale inou štruktúrou. **Riešením** môže byť explicitné zariadenie týchto **elementov** do rôznych **priestorov** mien a explicitný odkaz na tento priestor vo formáte **prefixu** mena:

```
<tituly xmlns:x=”http://www.w3.org/1999/xhtml”
      xmlns:k=”urn:knihy”>
  <x:titul>
    <x:tr>
      <x:td>Matematická analýza</x:td>
      <x:td>Jozef Novák</x:td>
      <x:td>122.50</x:td>
    </x:tr>
  </x:titul>
  <k:titul>
    <k:autor k:titul=”Ing” k:meno=”Jana Nováková”/>
    <k:názov>Technická dokumentácia</k:názov>
    <k:cena>95.00</k:cena>
  </k:titul>
</tituly>
```

Priestory mien **fungujú** na jednoduchom princípe – každý **element** aj atribút môže byť zaradený do **priestoru** mien, ktorý má svoju jednoznačnú **identifikáciu**. Úplná identifikácia prvku sa skladá z identifikácie menného priestoru a mena prvku. Aby identifikácia menného priestoru XML bola **unikátna**, používajú sa jednoznačné **označenia** pomocou URI referencií.

XIV. Uniformný identifikátor zdroja URI

URI je skratka pre uniformný **identifikátor** zdroja zavedený **pôvodne** pre iné účely, ale pre svoju jednoznačnosť využitý aj k **označeniu** menných priestorov. Referencie URI je postupnosť znakov v ACSII kóde, ktorá predstavuje **jednoznačný** identifikátor. Dve URI referencie sa považujú za **rovnaké**, ak sú tvorené identickou **postupnosťou** znakov.

Dve rôzne **URI** referencie môžu byť funkčne **ekvivalentné**, ale z hľadiska identifikácie sú rozdielne. URI referencia má predpísanú svoju **vnútornú** štruktúru – začína označením URI – schémy nasledovanej postupnosťou komponent. **URI** referencia je obvykle pomerne dlhá a môže obsahovať znaky, ktoré nesmú byť súčasťou mien v **XML dokumente**.

Preto sa využíva **zavedenie** zástupcu pre URI referencie pomocou atribútu xmlns. Takto zavedené meno potom zastupuje URI v **kvalifikovaných** identifikátoroch ako prefix. Mená prislúchajúce do menných priestorov XML sa **zapisujú** ako kvalifikované mená, t.j. ako dvojice **prefix**: **lokálna časť**.

Prefix určuje menný priestor, kvalifikované meno z menného priestoru zastúpeného prefixom html, ktorého lokálny **identifikátor** v rámci tohto priestoru je **head**. Ako prefixy nie je možné využívať mená začínajúce na **xml** alebo **xmlns**, pretože sú **rezervované**.

Pokiaľ neuvedieme **prefix** pred lokálne meno, chápe sa **rovnako** ako kvalifikované meno, kde menný priestor je **implicitný**. Nasledujúca ukážka by mala ilustrovať použitie **priestoru** mien. Je zjavné, ako sa zápis kráti, pretože pomerne dlhé URI je v **dokumente** uvedené iba raz.

```
<tovar xmlns:edi=”http://ecommerce.org/schema”>
  <polozka edi:daň=”oslobodene”>Jedlo pre
  deti</polozka>
</tovar>
```

XV. Štruktúra menného priestoru v XML

Menné priestory XML sa **líšia** od bežnej implementácie tohto pojmu v informatike v tom zmysle, že netvoria obyčajnú **množinu** v matematickom zmysle, ale majú svoju vnútornú štruktúru. Bežná interpretácia pojmu „**menný priestor**“ je kolekcia **identifikátorov**, ktorá neobsahuje **duplicity**.

Oproti tomu menný priestor XML sa všeobecne skladá z niekoľkých disjunktných množín:

- **oddiel mien elementov** – unikátne meno je dané menom priestoru a menom elementu,
- **oddiel globálnych atribútov** – unikátne meno je dané menom priestoru a menom atribútu,
- **oddiel lokálnych atribútov** pre každý element – unikátne meno je dané menom priestoru, menom elementu a lokálnym menom atribútu.

Inými slovami, môžu existovať elementy a globálne atribúty, ktoré sa volajú rovnako. Lokálne atribúty sú naviac rozlišované aj podľa svojich elementov.

Explicitný menný priestor

Nasledujúca ukážka **predstavuje explicitný** odkaz na menný priestor html. Prefix html je zavedený atribútom **xmlns** a zastupuje ako skratka jednoznačný **identifikátor** http://www.w3.org/1999/xhtml. Tento reťazec pripomína adresu **URL**, ale nie je to adresa stránky, iba identifikátor, ktorý si berie na pomoc **označenie** odkazu na stránky **W3C**.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<!—všetky elementy sú tu explicitne v priestore HTML
-->
<html:html     xmlns:html=”http://www.w3.org/TR/REC-
html40”>
  <html:head>
    <html:title>Dokument</html:title>
  </html>head>
  <html:body>
    <html:p>Skúste
    odkaz<html:a>href=”http>//document.com”>tu</html
    :a>
  </html:p>
</html:html>
```

Implicitný menný priestor

Ak uvedieme pri **niektorom** elemente atribút **xmlns** bez deklarácie prefixu, slúži ako implicitný menný **priestor** a nie je nutné ho **explicitne** uvádzať v menných vnorených elementoch či **atribútoch**.

Nasledujúci výpis **predstavuje** implicitný **odkaz** na menný priestor http://www.w3.org/1999/xhtml. Všetko, čo je obsahom **elementu** html patrí implicitne do menného priestoru určeného atribútom **xhtml** tohto **elementu**:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<!—všetky elementy sú tu implicitne v priestore
HTML -->
<html xmlns=”http://www.w3.org/1999/xhtml”>
  <head>
    <title>Dokument</title>
  </head>
  <body>
    <p>Skúste odkaz <a
    href=”http://deokument.com”>tu</a>.</p>
  </body>
</html>
```

Implicitný menný priestor dokumentu

Každý XML **dokument** má implicitne priradený **menný** priestor **XML**, ktorý je **identifikovaný** pomocou URI **referencie** http://www.w3.org/XML/1998/namespace, nesmie byť definované inak a toto URI nie je možné samozrejme použiť pre iný priestor. Ako **zástupca** má **zavedený** prefix xml.

Menný priestor **XML** obsahuje, okrem iného, tri **globálne atribúty** xml:lang, xml: space a xml:nil. Atribút xml:nil sa **používa** pre vyjadrenie **elementov** s prázdnym obsahom. Atribút xml:lang sa používa pre označenie **prirodzeného** jazyka, v ktorom je obsah **zapísaný**:

```
<odstavec xml:lang=”en”>The quick brown fox jumps
over the lazy dog.</odstavec>
<odstavec xml:lang=”en-GB”>What colour is
it?</odstavec>
<odstavec xml:lang=”en-US”>What colour is
it?</odstavec>
<verse kto=”Faust” cml:lang=”de”>
   <vers>Habe nun, ach! Philosophie, </vers>
   <vers>Juristerei, und Medizin</vers>
   <vers>und leider auch Theologie</vers>
   <vers>durchas studiert it heisem Bemu’n.</vers>
</verse>
```

A **konečne** atribút **xml:space** je možné použiť pre **vyjadrenie** požiadaviek na spracovanie bielych znakov. Jeho možné hodnoty sú default a **preserve**. Hodnota **preserve** vyjadruje, že všetky biele znaky majú zostať **zachované**. Hodnota default požaduje implicitný spôsob spracovávania podľa **kontextu**.

Prázdne URI

Pokiaľ **menný** priestor nedefinujeme, nepatria elementy do žiadneho menného priestoru. Nasledujúca ukážka **predstavuje** príklad prázdneho URI, kde **elementy** potom nie sú súčasťou žiadneho menného **priestoru**.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<! – všetky elementy nie sú v žiadnom priestore -->
<html xmlns=””>
  <head>
    <title>Dokument</title>
  </head>
  <body>
    <p>Skúste odkaz <a
    href=”http://dokument.com”>tu</a>.</p>
  </body>
</html>
```

Príklad použitia menných priestorov

Ukážme si príklad, kde **vytvárame** popis transformácie XML dokumentu v jazyku XSLT. Nasledujúca ukážka predstavuje **typické** použitie menných priestorov, kde pre mená z priestorov jazyka XSLT http://www.w3.org/1999/XLS/-transform je **zavedený** prefix xsl:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<xsl:stylesheet version=”1.0”
            xmlns:xls=”http://www.w3.org/1999/XSL/T
            ransform”>
  <xsl:output method=”xml” indent=”yes”/>
  <xsl:template match=”Fakulta”>
  <Fakulta>
     <xsl:value-of select=”@názov”/>
  </Fakulta>
    <xls:text> ----- </xls:text>
  </xls:template>
  </xls:stylesheet>
```

Aplikačné rozhranie

Aby mohli aplikácie s **XML** dokumentom jednoducho pracovať, využívajú štandardizované aplikačné rozhrania **XML** procesorov. **Existujú** štyri „**generické**“ API k procesorom, ktoré zároveň poskytujú istý dátový model XML, v ktorom sú XML dáta **reprezentované** ako strom:

- **SAX** – Simple API for XML (založené a udalostiach),
- **DOM** – Document Object Model (založené na stromoch a objektoch),
- **JDOM** – Java Document Object Model (založené na stromoch a objektoch),
- **StAX** – Streaming API for XML (založené na produktoch dát udalostiach).

Všetky **algoritmy** konštruujúce XML strom alebo **vracajúce** jeho uzly sú založené na sekvenčnom čítaní **XML textu**. Spomenuté rozhrania sú založené na modeli **Infoset**. Sekvenčné čítanie **XML** dokumentu odpovedá prechodom **príslušného** stromu do hĺbky vo variante **preorder**, dostávame uzly v poradí odpovedajúcom štandardom **usporiadania**.

XVI. Významová množina XML – Infoset

Ako už bolo povedané, **XML** dokument môže byť modelovaný ako strom alebo ako postupnosť udalostí, ktoré simuluje **prechod** stromom. Je nutné zdôrazniť, že špecifikácie významovej množiny niekedy využíva všeobecne **známe** pojmy, ako je práve **strom**.

V interpretácií týchto **pojmov** sa ale nemusí zhodovať s inými dátovými modelmi reprezentácie XML **dokumentov**. Pri spracovaní vstupného textu **XML** dokumentu sa predpokladá štandardná **normalizácia**. Pokiaľ nie je možné niektorú entitu nahradiť hodnotou, považuje sa za **neexpedovanú**.

Ak **potrebujeme** vyjadriť nedefinovanú hodnotu, používa sa kľúčové slovo **unknown**, pretože označenie **null** má príliš veľa iných interpretácií. **Infoset** je novším modelom než model **XML** dát použitý v jazyku **XPath**. Nie všetky **XML** technológie ho preto **používajú**.

Každá významová množina **XML** dokumentu obsahuje práve jednu **položku** typu popis dokumentu. Všetky ostatné informačné položky sú **dostupné** z tejto položky – **priamo** alebo **nepriamo**. Pokiaľ významová množina neobsahuje popis **dokumentu**, jedná sa o významovú množinu nejakého **fragmentu XML** dokumentu.

Popis dokumentu **obsahuje** odkazy na usporiadaný zoznam potomkov – radenie je dávno štandardným usporiadaním položiek dokumentu. **Zoznam** obsahuje práve jeden koreňový element, pretože sa jedná o reprezentáciu stromu. Môže ale **obsahovať** ďalšie inštrukcie pre spracovanie a komentáre, ktoré sa **vyskytujú** mimo koreňový **element**.

```
<?xml version="1.0" encoding="iso-8859-2"?>
<sprava>
  <pozdrav farba="modrá">Ahoj, svet!</pozdrav>
</sprava>
```

Jeho významová **množina reprezentovaná** ako strom obsahuje za celý dokument **položku**:

```
Položka D1:
node-kind(D1) = "document"
children(D1) = (E1)
document-element(D1) = "správa"
base-URI = "http://www.w3.ogr/XML/1998/namespace"
character-encoding-scheme(D1) = "iso-8859-2"
version(D1) = "1.0"
```

Pre každý element **dokumentu** obsahuje významová množina práve jeden popis elementu. Významová množina dokumentu správa bude **obsahovať** za elementmi správa a pozdrav **položky**:

```
Položka E1:
node-kind(E1) = "element"
local-name(E1) = "správa"
children(E1) = (E2)
attributes(E1) = ()
parent(E1) = (D1)

Položka E2:
node-kind(E2) = "element"
local-name(E2) = "pozdrav"
children(E2) = (E2)
attributes(E2) = (A1)

parent(E2) = (E1)
```

Pre každú **inštrukciu** obsiahnutú v dokumente obsahuje významová množina práve jeden popis inštrukcie. Pre každú **neexpandovanú** referenciu na externú entitu, ktorú XML procesor pri spracovávaní dokumentu odhalí, obsahuje **významová** množina práve jeden popis neexpandovanej **referencie** na **entitu**.

Môže poslúžiť ako **označenie** miesta, kde nejaká informácia chýba. Pokiaľ je dokument validný, nemala by sa v ňom táto **položka** objaviť. Pre všetky **znakové** položky obsiahnuté v dokumente obsahuje významovú množinu pre každý znak **práve** jeden popis **znakovej položky**.

Významová množina dokumentu správa bude obsahovať za textový obsah "Ahoj, svet!" element pozdrav znakovej **položky**:

```
Položka C1:
node-kind(C1) = "character"
character-code(C1) = 'A'
parent(C1) = (E2)


Položka C2:
node-kind(C2) = "character"
character-code(C2) = 'h'
parent(C2) = (E2)
…
Položka C12:
node-kind(C12) = "character"
character-code(C12) = '!'
parent}C12| = (E12)
```

Aj napriek tomu, že v definícií modelu **InfoSet XML** dokumentov je každý znak reprezentovaný **samostatnou** položkou, môžu **XML** procesory predávať aplikáciám celé reťazce Pre každý **komentár** obsiahnutý v dokumente obsahuje významová množina práve jeden popis **komentára**.

Pokiaľ má XML dokument určené **DTD**, potom pre každé DTD odkazované v **dokumente** obsahuje významová množina práve jeden popis **DTD**, ktorý vznikne zlúčením všetkých odkazovaných DTD. Pri **zlučovaní** majú prednosť prvé deklarácie pred **nasledujúcimi**.

Pre každú **neanalyzovanú** entitu v XML dokumente obsahuje významová množina práve jeden popis neanalyzovanej **všeobecnej** entity, deklarovanej v DTD. Na rozdiel od popisu **neexpandovanej** referencie na entitu sa tu jedná o definíciu entít, ktoré neboli **spracované** behom analýzy vstupného **dokumentu**, pretože sú napr. určené iné **aplikácie**.

Pre každú **notáciu** uvedenú v DTD obsahuje **významová** množina popis notácie. Ten sa rovnako ako popis entity stáva súčasťou **reprezentácie** dokumentu, nie však reprezentácie DTD, v ktorom bol definovaný. Pre každý menný **priestor** zavedený v dokumente obsahuje **významová množina** práve jeden popis menného **priestoru**.

Každý XML dokument má **definované poradie** položiek, ktoré obsahuje, štandardné usporiadanie položiek dokumentu. Toto **usporiadanie** nie je v rámci jedného spracovávania elementu dané normou, ale **implementáciou**.

```
<?xml version = “1.0”?>
<zrp:sprava dok:datum=”19990421”
        xmlns:dok=”http://priklad.org/namespaces/dok”
        xmlns:zpr=”http://spava.priklad.org”/>
Ahoj svet!</zrp:sprava>
```

Všetky položky **významovej** množiny si môžeme predstaviť ako uzly stromu. Koreň stromu predstavuje položka typu **dokument**. Hierarchické usporiadanie v strome je dané vlastnosťami [children] a [parent]. Štandardné **poradie** položiek je dané **prehľadávaním** stromu do hĺbky.

Prvú položku v poradí predstavuje vždy koreň stromu, t.j. položka typu dokument. Poradie súrodencov v strome je dané ich poradím v **reprezentovanom** XML dokumente. Uzly typu menný priestor bezprostredne **nasledujú** element, ku ktorému sa vzťahujú. Ich **vzájomné** poradie je **implementačne** závislé.

Uzly typu atribút sú **zaradené** za prípadné uzly typu menný priestor, ich vzájomné poradie je opäť implementačne závislé. Pre **prechádzanie XML** stromu je možné definovať rozmanité **traverzovacie** funkcie. Tie umožnia aplikáciám prístup k **položkám** XML **dokumentu**.

Významová **množina** skôr uvedeného XML dokumentu so správou môže byť vyjadrená nasledujúcim **serializovaným** vyjadrením XML stromu, ktoré je možné **zapísať** pomocou výsledkov **traverzovaných** funkcií:

```
// Document node D1
dm:node-kind(D1) = “document”
dm:string-value(D1) = ([E1])

// Element node E1
dm:node-kind(E1) = “element”
dm:node- name(E1) = xs:QName(””,”správa”)
dm:string-value(E1) = “Ahoj, svet!”

dm:type (E1) = xs:anyType
dm:parent(E1) = ([D1])
dm:children (E1) = (E2)
dm:attributes(E1) = ()

//Element node B2
dm:node-kind(E2) = “element”
dm:node-name(E2) = xs:QName(““,“pozdrav“)
dm:string-value(E2) = “Ahoj, svet!”
dm:type(E2) = xs:anyType
dm:parent(E2) = ([E1])
dm:children(E2) = ()
dm:attributes(E2) = ([A1])
```

```
// Attribute node A1
dm:node-kind(A1) = “attribute”
dm:node-name = xs:QName(“”, ”farba”)
dm:string-value(A1) = “modrá”
dm:type-value(A1) = “modrá”
dm:type(A1) = xs:string
dm:parent (A1) = ([E2])

// Text node T1
dm:node-kind(T1) = “text”
dm:string-value(T1) = “Ahoj, svet!”
dm:type-value(T1) = xs:anySimpleType(“Ahoj, svet!”)
dm:type(T1) = xs:anySimpleType
dm:parent (T1) = ([E2])
```

Uveďme si pre ilustráciu prehľad **vlastností** XML dokumentu, ktoré sa do jeho významovej množiny nepremietnu – nemusí teda byť pomocou **XML** procesoru **zistiteľné**.

- Definícia **obsahu elementu** z DTD, ako bola zavedená konštrukciou **ELEMENT**.
- Poradie **deklarácie** atribútov v DTD, ako bola zavedená konštrukciou **ATTLIST**.
- Meno **typu dokumentu**, ako bolo zavedené konštrukciou **DOCYTYPE**.
- **Biele znaky**, ktoré sa nachádzajú **mimo** element dokumentu.
- **Biele znaky**, ktoré sa nachádzajú **bezprostredne** za menom aplikácie v inštrukciách
- Nie je možné **rozpoznať poradie**, v ktorom boli uvedené hodnoty atribútu v **otváracej** zátvorke **elementu**.

- Nie je možné **rozpoznať poradie** deklarácie v **DTD** apod.

Stručnejšie, významová množina dokumentu neobsahuje žiadne informácie z DTD. Významová množina **dokumentu** rovnako nezávisí na spôsobe jej zostavenia, závisí iba na **normalizovanom obsahu** dokumentu.

Spracovanie XML pomocou SAX

Rozhranie SAX predstavuje v **spracovávaní** XML de facto štandard. Ide o rozhranie založené na udalostiach, t.j. situáciách, kedy sa pri **sekvenčnom** čítaní spracovávaného dokumentu **narazí** napr. na:

- **začiatok dokumentu**,
- **počiatočnú značku elementu**,
- **koncovú značku element**,
- **znakové dáta**,
- **inštrukcie pre spracovávanie**.

SAX informuje o **udalostiach**, kedykoľvek vidí uzol pre značku, atribút, text, externú entitu. Ak narazíme na element, vracia tento **element**, zoznam jeho atribútov a obsah. SAX sám o sebe nie je XML procesorom. **Programátor** pripája vlastné funkcie pre používanie udalosti. Každý udalosť teda vyvolá **korešpondujúcu funkciu**, ktorou píše programátor.

Tento spôsob spracovania je teda prúdovo **orientovaný**. Spracovávanie dáva výsledky bez toho, aby bol k dispozícii celý **XML dokument**.

```
<?xml version = “1.0”?>
<kniha trieda=”H.3.3”>
  <autor>Adam Šangala</autor>
  <titul>XML pre všetkých</titul>
  <kapitola>
     <nadpis>Úvod</nadpis>
     Táto kniha sa...
  </kapitola>
…
</kniha>
```

Pre uvedený **XML** dokument je generovaná **nasledujúca** postupnosť udalostí:

1. **startDocument**() – ohlásenie začiatku dokumentu,
2. **startElement**():kniha – dáva značku a atribúty,
3. **startElement**(): autor,
4. **characters**(): Adam – procesor bude volať túto metódu pre každú časť znakových dát, aké časti textu to budú však závisí na konkrétnej implementácií parseru,
5. **characters**(): Kosek,
6. **endElement**(): autor
7. **startElement**(): titul,
8. ...
9. **endDocument**() – ohlásenie konca dokumentu.

Procesor **založený** na tomto spôsobe spracovania je orientovaný na dáta. Základnou myšlienkou je **spracovať** element a potom ho „zabudnúť“. Zaujímavou otázkou je ako využiť SAX pre implementáciu **opytovacích** jazykov nad XML dátami. Spôsob prechodu stromom daný štandardným usporiadaním má zrejme vplyv na **spracovanie** otázok.

Napr. nie je možné sa vracať do **susedných** vetiev stromu. Pri štúdiu typu otázok v jazyku **XPath** ľahko zistíme, že so **SAX** je možné efektívne **implementovať** iba otázky odpovedajúce jednoduchým cestám v XML **strome**.

Spracovanie XML pomocou DOM

DOM je projekciou XML **Infosetu**. Objektový model DOM reprezentuje **Infoset** ako strom **uzlov.**

Uzol DOM	Informačná položka Infoset
Document	dokument
DocumentFragment	neaplikovateľné
EntityReference	odkaz na nenahradenú entitu
Element	element
Attr	atribút
ProcessingInstruction	inštrukcie pre spracovanie
Comment	komentár
Text a CDATASection	postupnosť znakových informačných prvkov
DocumentType	DTD
Entity	neparsovaná entita
Notation	notácia

Tab. 2 Informačné prvky INFOSET a uzly DOM

DOM poskytuje **objektovo** orientované rozhranie závislé na platforme a jazyku. Jazykom špecifikácie DOM je ODM-IDL. Pri **spracovaní** XML dát generuje procesor vo vnútornej pamäti strom, ktorý odpovedá spracovávanému **dokumentu**. Rozhranie DOM definuje metódy pre prístup a tiež modifikácií stromu. Medzi spoločné **metódy** patrí napr.:

```
Node.getNodeType()
Node.getNodeName()
Node.getFirstChild()
Node.getLastChild()
Node.getNextSibling()
Node.getPreviousSibling()
Node.getAttributes()
```

Procesor **založený** na tomto type spracovania je orientovaný na dokument. Na rozdiel od SAX je možné na DOM založiť aj jednoduchú **implementáciu jazykov** ako XPath či jQuery.

Spracovanie XML pomocou JDOM

Standard DOM je veľmi **jednoduchá** dátová štruktúra, v ktorej sa miešajú textové uzly, uzly elementov, uzly inštrukcií pre **spracovanie** a iné. To znepríjemňuje použitie DOM v praxi. JDOM na druhej strane pre XML dáta **vytvára** strom objektov zodpovedajúcich typu, t.j predstavuje objektovo **orientované** rozhranie určené špeciálne pre jazyk **java**.

Procesor opäť generuje vo vnútornej pamäti strom zodpovedajúci vstupnému dokumentu, ktorý je pre použitie **jednoduchšie**. JDOM rozhranie **samozrejme** zahŕňa aj DOM metódy pre prístup a **modifikáciu** tohto **stromu**.

Spracovanie XML pomocou StAX

Typickým rysom SAX API je „**pretláčanie**“ informácií o XML aplikácií, ako náhle na ňu narazí a to bez ohľadu na to, či ju aplikácia potrebuje alebo nie. Na **programátorskej** úrovni spočíva tento princíp v tom, že vývojár **umiestni** do aplikácie procedúry s kódom, ktorý bude reagovať na **príslušné udalosti**.

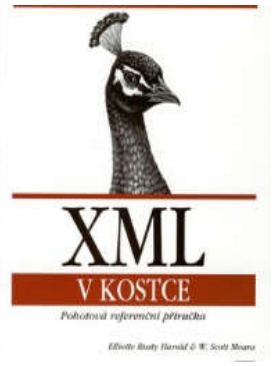
Naproti tomu existuje **ďalšie** kategórie prúdových API, ktoré sa nazývajú pull-API. V týchto API skôr klientsky program žiada XML **procesor** o ďalšie informácie , t.j. API je **riadené** klientom a nie **naopak**.

Proces syntaktickej **analýzy** je riadený žiadosťou o novú **udalosť** na rozdiel od klasickej SAX analýzy, kedy sú udalosti postupne generované tak, ako je **načítaný** dokument. Predstaviteľom pull-API je StAX. StAX **implementuje** dve možnosti práce s XML dokumentmi: **kurzorové** a **udalostné**.

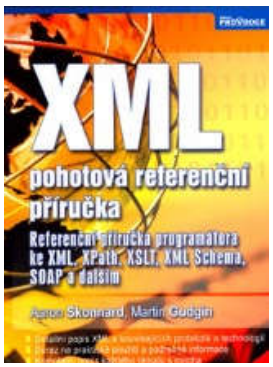
Primárne je **využívané** nízkoúrovňové kurzorové rozhranie, ktoré v sekvenčnom pohybe vždy ukazuje v danom čase na jednu vec. **Pomocou** metód ako napríklad next() alebo hasNext() umožňuje StAX žiadať o **nasledujúcu** udalosť skôr ako pracovať s udalosťou vo vracanom kóde ako SAX. Celý dokument je **prechádzaný** v jednom **while** cykle.

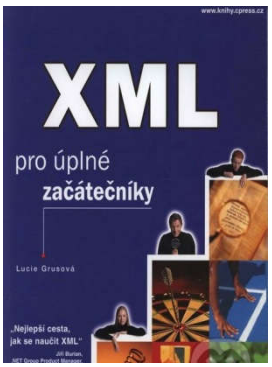
XVII. Odporúčaná literatúra a zdroje

1. **XML technologie –** L. Mlýnková, M. Nečaský, J. Pokorný
2. **XML začíname programovat –** Miroslav Žák
3. **XML v kostce -** Elliotte Rusty Harold, W. Scott Means
4. **XML krok za krokem –** Michael J. Young



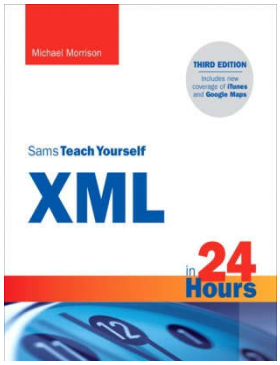
5. **XML – efektivní programování -** Dino Esposito
6. **XML pro každého –** Jiří Kosek
7. **XML pohotová referenční príručka –** A. Skonnard, M. Gudgin
8. **XML pro úplné začátečníky -** Lucie Grusová

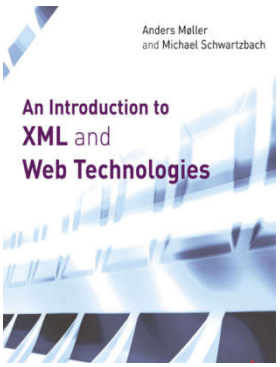


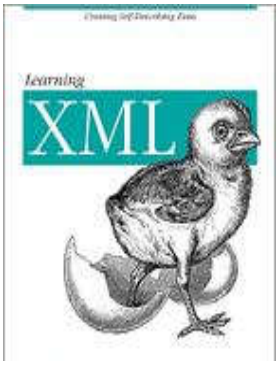


Zahraničná literatúra

1. **XML second edition –** Kevin Howard Goldberg
2. **Teach yourself XML in 24 hours –** Michael Morisson
3. **Learning XML –** Erik T. Ray
4. **An introduction to XML and Web Technologies –** Anders Moller







5. **XML in a Nutshell –** Elliotte Rusty Harold
6. **The XML schema complete refernce –** Cliff Binstock
7. **Definitive XML schema –** Priscilla Walmsley
8. **Inside XML –** Steven Holzner