# Stručný obsah

# Obsah

# Předmluva

Když jsem před více než deseti lety psal předmluvu k dnes již legendární knize *PHP – tvorba interaktivních internetových aplikací* [11], slíbil jsem v ní, že na doprovodný web knihy umístím informace o práci s formátem XML, které se do knihy již nevešly. Z časových důvodů k naplnění tohoto slibu nikdy nedošlo. Jako satisfakci proto přijměte tuto knihu. Její náplní je pouze XML a jeho použití v PHP.

Deset let je dlouhá doba, ale myslím, že čekání se vyplatilo. Podpora XML byla v PHP až do jeho verze 5.0 poměrně partyzánská. A teprve od verze 5.2 lze PHP považovat za jazyk, ve kterém se dá s XML rozumně pracovat.

O významu XML dnes již není potřeba nikoho přesvědčovat. Ať se nám to líbí nebo ne, XML je zkrátka všude a ve webových aplikacích je potřeba s tímto formátem pracovat. Ať už se jedná o importy a exporty dat, transformaci dat pro prezentační vrstvu nebo backend pro AJAXovou aplikaci. Cílem této knihy je naučit vás používat všechna existující rozhraní pro práci s technologiemi XML, která PHP nabízí. Popsán a vysvětlen je však i samotný jazyk XML, jeho syntaxe a navazující technologie jako XML schémata, dotazovací jazyk XPath a transformační jazyk XSLT. Pro pochopení výkladu tak není nutná žádná velká předchozí zkušenost s XML – vše potřebné je průběžně vysvětleno.

V dnešní době začínají být papír a knihy považovány za příliš konzervativní médium. Asi jsem staromilec a mám tištěné knihy rád. Nicméně na adrese *http://www.kosek.cz/knihy/phpxml/* najdete další informace související s knihou – příklady ke stažení, opravy chyb apod. Máte-li ke knize nějaké připomínky, uvítám je na mojí emailové adrese `<jirka@kosek.cz>`.

Na vzniku knihy má zásluhu mnoho lidí. Nevyčerpatelnou trpělivost prokázali šéfredaktoři počítačové redakce Miroslav Lochman a Daniel Vrba, kteří vydrželi pět let čekat na dokončení knihy. Pokud v knize nebude příliš chyb, je to zásluha korektorky Zuzany Vrbové a redaktorky Evy Steinbachové. Největší dík však patří mé ženě Lence a dětem, kteří trpělivě snášely mé útěky k počítači při psaní knihy.

Přeji vám příjemné čtení knihy.

*Jirka Kosek*

Liberec, 10. dubna 2009

# Úvod



Vývoj moderních webových aplikací klade na vývojáře vysoké nároky, je potřeba znát široké spektrum technologií. Počínaje jazyky HTML a CSS pro definici samotné stránky a jejího vzhledu, přes Javascript pro vytváření vysoce interaktivních aplikací, po nějaký serverový jazyk jako je PHP. Každá větší aplikace navíc potřebuje někam ukládat data, typicky do databáze. K tomu je potřeba znát principy protokolu HTTP a vědět, jak obcházet jeho limity. A aby toho nebylo málo, na mnoha místech se vývojář webové aplikace setká i s formátem XML.

Když XML v polovině 90. let minulého století vznikalo, původní smělé plány byly, že zcela nahradí jazyk HTML při doručování obsahu do prohlížeče. Tato myšlenka se však ukázala jako příliš revoluční a navíc problémy spojené s jazykem XHTML a jeho podporou v prohlížečích v očích mnoha webových vývojářů nevrhly na XML příliš růžové světlo. Nicméně technologie XML jsou dnes pevnou součástí mnoha webových technologií, formátů a protokolů, takže je potřeba práci s tímto formátem ovládat.

Kde se na webu s XML může vývojář setkat? Syntaxi XML využívají mnohé prezentační formáty – počínaje jazykem XHTML, přes stále populárnější vektorový grafický formát SVG, až po jazyky pro definici uživatelského rozhraní v moderních RIA (Rich Internet Application) prostředích, jako je XAML v Silverlightu a MXML ve Flashi. Pokud tedy vaše skripty v PHP v minulosti generovaly převážně HTML kód, časem budou přicházet požadavky na dynamickou tvorbu modernějších, na XML založených, formátů.

XML dnes zcela dominuje na poli publikování metainformací. Jedná se například o formáty pro publikování přehledů nových článků, jako jsou RSS či Atom. Protože začleňování sémantiky ve strojově čitelné podobě přímo do webových stránek je stále v plenkách, mnoho vyhledávačů nabízí vlastní formáty, ve kterých jim můžete předávat informace vylepšující vyhledávání – například Google Sitemap nebo Google Base.

Další využití XML je pro komunikaci a předávání dat. XML se využívá jednak pro výměnu dat mezi backendy jednotlivých aplikací a dále pak v AJAXových aplikacích pro zasílání aktualizací dat do prohlížeče. „Enterprise“ aplikace pak pro samotnou komunikaci nevyužívají prosté XML, ale komplexnější mechanismus webových služeb.

Diverzita koncových zařízení používaných pro přístup k webu se také stále zvyšuje. Webové stránky se už neprohlížejí jen z klasického počítače, ale i z chytrých telefonů nebo různých PDA. Mnohé aplikace potřebují uspokojivě řešit možnost kvalitního tisku. Už nestačí vytvořit aplikaci, která výstupy generuje jen jako HTML. Pro jednotlivá koncová zařízení je potřeba generovat odlišné formáty výstupu a webovou aplikaci je potřeba obohatit o flexibilní prezentační vrstvu. Pro vytvoření takové vrstvy lze využít i jazyk XML a stylové technologie jako XSL.

Výše uvedený výčet toho, kde se na webu můžeme setkat s XML, jistě není úplný. Pouze potvrzuje to, že webový vývojář se dnes neobejde bez znalosti tohoto formátu a práce s ním. Tato kniha vás naučí vše potřebné o formátu XML a navazujících technologiích a o tom, jak lze s tímto formátem pracovat v PHP.

První kapitola je určená zejména pro čtenáře, kteří ještě neznají formát XML. Seznámí se zde se syntaxí jazyka a naučí se ji kontrolovat.

Druhá kapitola pak stručně shrnuje a ukazuje, jaké možnosti pro práci s XML nabízí PHP. Je to ideální místo pro porovnání jednotlivých přístupů pro práci s XML. Nemusíte tak číst celou knihu, ale v této kapitole zjistíte, jaký způsob práce s XML je pro vás nejvhodnější a ten si dále podrobněji nastudujete v odpovídající samostatné kapitole.

Třetí kapitola přímo nesouvisí s XML, ale ukazuje, jak lze v PHP částečně obejít chybějící podporu znakové sady Unicode, kterou využívá i jazyk XML.

Následují čtyři kapitoly, které podrobně popisují jednotlivá rozhraní pro práci s XML – SimpleXML, SAX, DOM a XMLReader. Výběr vhodného rozhraní záleží na povaze dat, která čtete, a na tom, jak je potřebujete zpracovat.

Osmá kapitola seznamuje s dotazovacím jazykem XPath, který nabízí jednoduchou a zároveň mocnou metodu pro vyhledávání a výběr dat v dokumentech XML. Kromě samotného dotazovacího jazyka je zde samozřejmě popsáno, jak jej používat v kombinaci s rozhraními DOM a SimpleXML.

Další dvě kapitoly se věnují kontrole (validaci) dokumentů XML. Zvláště v otevřeném prostředí internetu je potřeba počítat s nejhorším a všechny vstupy do aplikace pečlivě kontrolovat. Pro dokumenty XML takovou kontrolu nabízejí schémata, která dokáží popsat povolenou strukturu a datové typy dokument XML. Devátá kapitola tak popisuje nejpoužívanější schémové jazyky a v desáté kapitole je pak ukázáno, jak pomocí nich prakticky v PHP kontrolovat data uložená v XML.

Jedenáctá kapitola vysvětluje základy jazyka XSLT a jeho použití v PHP. XSLT je nejvhodnější prostředek pro transformace dokumentů XML do dalších formátů, včetně formátu HTML. Nalezne tak uplatnění například v prezentační vrstvě webové aplikace.

Následující dvanáctá kapitola se pak věnuje komunikaci mezi aplikacemi s využitím XML. Popsány jsou jak klasické webové služby, tak jednodušší mechanismy jako REST a AJAX.

Poslední třináctá kapitola pak ukazuje možnosti pro generování dokumentů XML na výstupu skriptu.

Kniha je zaměřena zejména na vysvětlení principů a na ukázky využití jednotlivých technologií a knihoven PHP. Ve většině případů jsou popsány všechny možnosti jednotlivých knihoven PHP. Nicméně kniha primárně neslouží jako referenční příručka, pro tyto účely je vhodné nahlížet i do dokumentace PHP na adrese *http://docs.php.net/manual/en/*.

Všechny příklady byly testovány s PHP ve verzi 5.2 a v současné době nic nenasvědčuje tomu, že by se v blízké době mělo v jazyce PHP měnit něco, co by způsobilo nefunkčnost skriptů. Většina použitých knihoven je standardní součástí PHP. Chcete-li používat XSLT, je potřeba do PHP přidat modul `php_xsl`, pro webové služby je zapotřebí modul `php_soap` a kapitola o Unicode využívá některé funkce z modulu `php_mbstring`.

Dlouhé řádky ve výpisech, které musely být rozděleny, jsou označeny pomocí znaku ‚►‘.

# 1.



# Syntaxe XML

Chceme-li pracovat s dokumenty XML, musíme samozřejmě vědět, jak tyto dokumenty vypadají. V této kapitole se proto seznámíme se syntaxí jazyka XML a dalšími jeho rysy, které bychom měli znát. Znáte-li již XML dobře, a zajímá vás jen, jak se s ním pracuje v PHP, můžete tuto kapitolu směle přeskočit.

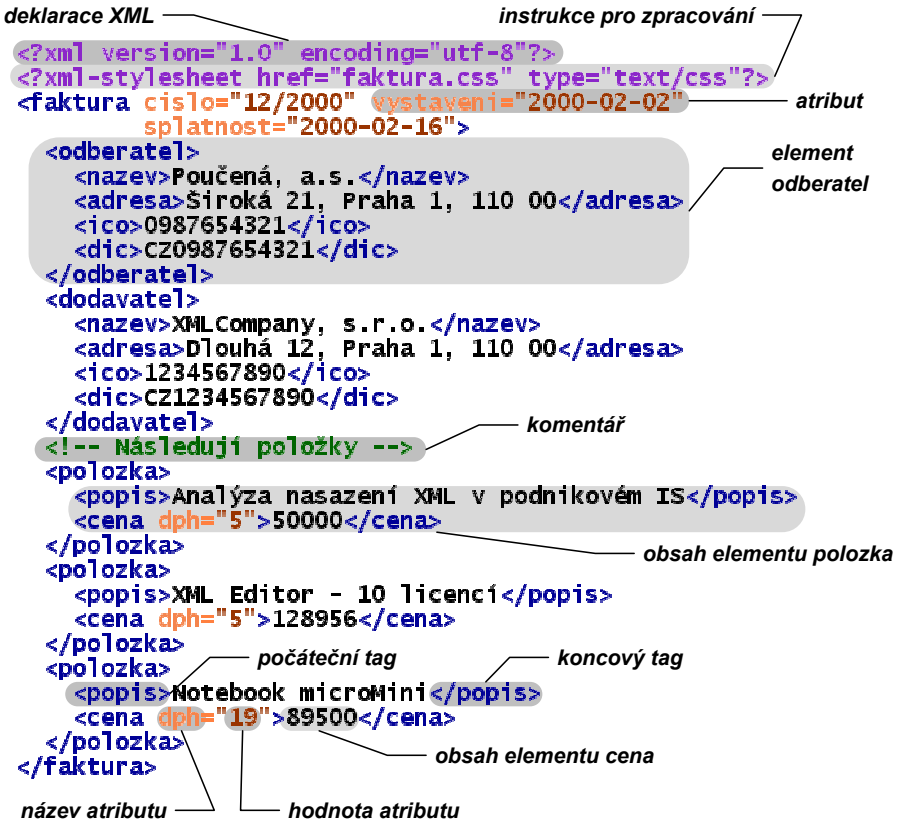
## 1.1 Elementy a struktura dokumentu

Každý XML dokument se skládá z *elementů*. Elementy se v textu vyznačují pomocí tzv. *tagů*. Většině elementů odpovídají dva tagy – počáteční a koncový.

```
<para>Toto je obsah elementu para.</para>
```

Ukázka obsahuje jeden element `para`. Jeho obsah je vyznačen pomocí tagů `<para>` (počáteční tag) a `</para>` (ukončovací tag). Jen na okraj poznamenejme, že výše uvedená ukázka je asi nejjednodušším dokumentem XML, který vůbec můžeme vytvořit.

Názvy tagů se zapisují mezi znaky ‚<' a ‚>'. Ukončovací tag má před svým názvem ještě znak ‚/', aby se odlišil od počátečního.




*Obrázek 1.1: Základní části dokumentu XML*

Některé elementy nemusejí mít žádný obsah. Můžeme je samozřejmě zapisovat tak, že za počátečním tagem uvedeme hned ten koncový.

```
<para>Toto je obsah elementu para.<br></br> A tohle také.</para>
```

Není to však příliš pohodlné, a proto můžeme v XML použít ještě jednu variantu tagu, která říká, že element nemá žádný obsah. Počáteční tag ukončíme dvojicí znaků ‚/>' místo pouhého ‚>' a koncový tag vynecháme.

```
<para>Toto je obsah elementu para.<br/> A tohle také.</para>
```

Každý dokument XML musí obsahovat pro všechny počáteční tagy odpovídající koncový tag, nebo musí být počáteční tag zapsán jako element s prázdným obsahem. Následující ukázky jsou ukázkami špatných dokumentů, které nevyhovují specifikaci XML.

```
<para>Toto je obsah elementu para.<br> A tohle také.</para>
```

Ukázka je chybná, neboť tag `<br>` není ukončen.

```
<para>Toto je obsah elementu para.<br/> A tohle také.</oara>
```

Počáteční tag <para> není ukončen a k ukončovacímu tagu </oara> v dokumentu neexistuje odpovídající počáteční tag. Chybou rovněž je, když se elementy v dokumentu kříží.

```
<b>Ukázka <i>překřížení</b> elementů</i>
```

Každý dokument XML musí být celý obsažen v jednom elementu. Následující ukázka tedy nepředstavuje správný dokument XML.

```
<nadpis>Pokusný nadpis</nadpis>
<odstavec>První odstavec</odstavec>
<odstavec>Druhý odstavec</odstavec>
<odstavec>Třetí odstavec</odstavec>
```

Stačí však přidat jeden element, který vše „obalí“, a vše je v pořádku.

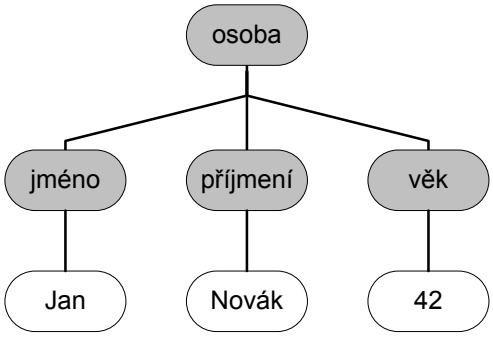
```
<článek>
  <nadpis>Pokusný nadpis</nadpis>
  <odstavec>První odstavec</odstavec>
  <odstavec>Druhý odstavec</odstavec>
  <odstavec>Třetí odstavec</odstavec>
</článek>
```



# 1.2 Datový model dokumentu

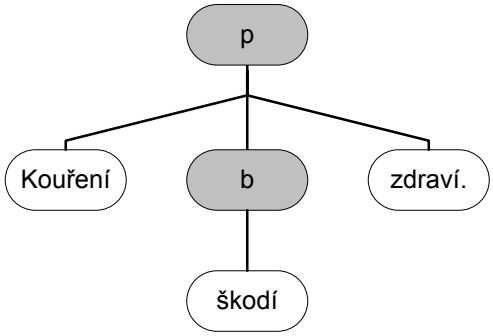
Viděli jsme, že elementy můžeme do sebe zanořovat, takže element může obsahovat další elementy nebo text. Elementy tak vytvářejí hierarchickou stromovou strukturu. Každý dokument XML si proto můžeme představit jako strom, jehož jednotlivé uzly odpovídají jednotlivým elementům (šedé uzly v obrázku), případně textu uvnitř elementů (bílé uzly v obrázku).

```
<osoba>
  <jméno>Jan</jméno>
  <příjmení>Novák</příjmení>
  <věk>42</věk>
</osoba>
```



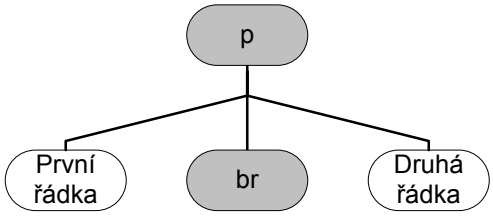

Uzly odpovídající textovému obsahu elementů jsou ve stromu vždy na nejnižší úrovni listů a už na ně nemohou být navěšeny žádné další uzly. V případě, že má element tzv. *smíšený obsah*, jsou jeho dětmi ve stromové reprezentaci jak textové uzly, tak uzly odpovídající elementům.

```
<p>Kouření
<b>škodí</b> zdraví.</p>
```




Prázdné elementy se ve stromu dokumentu objeví jako uzly, které už také nemají žádné děti.

```
<p>První řádka<br/>
Druhá řádka</p>
```




Výše popsanému stromovému modelu dokumentu XML se říká *infoset* [7]. Abstraktní datový model infosetu stručně řečeno říká, že dokument XML je stromová struktura složená z jednotlivých uzlů. Uzlů je přitom několik typů (elementy, textové uzly, atributy, komentáře, instrukce pro zpracování, jmenné prostory a další). U každého uzlu pak infoset definuje několik jeho vlastností jako jméno, rodiče, seznam dětí apod. Na infosetu je tak založena většina jazyků a rozhraní, které jsou vystaveny nad XML. Je to pochopitelné, protože při práci s dokumentem XML nás většinou zajímá jeho struktura a obsah zachycený v elementech, a infoset nabízí právě tento pohled na dokument XML. Většinou nás totiž nezajímá pohled na dokument XML jako na textový soubor, ve kterém se vyskytují speciální značky oddělené pomocí znaků ‚<' a ‚>' od ostatního textu, protože bychom se museli sami starat o syntaktickou analýzu takového zdrojového textu.

# 1.3 Atributy

Elementy jsou základním prostředkem pro členění informací uvnitř dokumentu XML. Kromě elementů lze pro zachycení informací využít *atributy*. Atributy se vždy zapisují k počátečnímu tagu elementu.

```
<odstavec zabezpečení="důvěrné">Nějaká tajná informace.</odstavec>
```

V naší ukázce jsme atributu `zabezpečení` přiřadili hodnotu `důvěrné`. Hodnotu atributu musíme vždy uzavřít do uvozovek nebo do apostrofů. U jednoho tagu lze použít více atributů najednou, stačí je oddělit mezerou.

```
<odstavec zabezpečení="důvěrné" autor='Jan Novák'>Nějaká tajná informace.</▶
odstavec>
```

U jednoho elementu se přitom nemohou použít dva atributy se shodným názvem. V mnoha případech je jedno, zda nějakou informaci uložíme jako element, nebo atribut. Srovnejte například následující dva fragmenty dokumentu XML:

```
<osoba věk="42">
  <jméno>Pepa</jméno>
</osoba>

<osoba>
  <jméno>Pepa</jméno>
  <věk>42</věk>
</osoba>
```

Nicméně z praxe postupně vyplynulo několik pravidel, která vám pomohou vybrat si, zda je v dané situaci lepší použít element nebo atribut. S pravidly se seznámíme v části 9.6.1.



# 1.4 Zápis vyhrazených znaků

Vzhledem k tomu, že se znak ‚<' používá pro oddělení tagů od okolního textu, není možné jej zapsat do dokumentu jen tak. Musíme ho opsat jako znakovou entitu `<`.

```
Vyřešte nerovnost 3x &lt; 5
```

Vidíme, že odkaz na znakovou entitu začíná znakem ‚`&`', proto i tento znak musíme do dokumentu vkládat opisem `&`.

```
Křupavé rohlíčky vám dodá pekařství Žemlička &amp; syn
```

Pokud potřebujeme uvnitř hodnoty atributu použít zároveň uvozovky i apostrofy, s výhodou využijeme odpovídající opisy `&quot;` a `&apos;`. XML definuje ještě pátou znakovou entitu `>`, která zastupuje znak ‚>'. Tento znak však ve většině případů můžeme zapisovat přímo bez nutnosti opisu.[1]

*Tabulka 1.1: Předdefinované znakové entity XML*

| Entita | Znak |
|---|---|
| `<` | < |
| `&` | & |
| `>` | > |
| `&apos;` | ' |
| `&quot;` | " |

Jazyk XML definuje pouze těchto pět entit. Další znakové entity, které známe z HTML, jako například ` `, `&copy;` nebo `&ndash;`, v XML k dispozici nejsou, nicméně si je můžeme v případě potřeby nadefinovat sami (viz 1.13.1.1).

[1] Opis je nutný pouze v případě, že by se v dokumentu vyskytovala sekvence znaků ‚`]]>`'. Ta se musí přepsat jako `]]>`.

# 1.5 Názvy elementů a atributů

XML je (na rozdíl například od HTML) citlivé na velikost písmen. Počáteční a koncový tag se proto musí shodovat i ve velikosti písmen. Následující element je chybný, protože si neodpovídá počáteční a koncový tag:

```
<NÁZEV>Tento element je zapsán špatně.</název>
```

Samotná jména elementů a atributů mohou přitom být vytvářena poměrně volně. První znak jména musí být písmeno nebo podtržítko, další znaky mohou navíc obsahovat i čísla, tečku a pomlčku. Písmena přitom mohou být i z jiné než anglické abecedy. Jména elementů a atributů tak můžeme psát klidně česky, nebo třeba rusky v azbuce:

```
<имя>Булгаков</имя>
```

# 1.6 Deklarace XML

Každý dokument XML by měl začínat deklarací XML, ve které určíme, jakou verzi XML používáme a v jakém kódování je soubor uložen.

```
<?xml version="1.0" encoding="utf-8"?>
<osoba>
  <jméno>Jan</jméno>
  <příjmení>Novák</příjmení>
  <věk>42</věk>
</osoba>
```

Každá aplikace, která podporuje XML, musí umět zpracovat soubor uložený v kódování UTF-8 nebo UTF-16. Proto bychom měli dokumenty XML přednostně ukládat a ostatním posílat v jednom z těchto kódování. V praxi se přitom častěji používá UTF-8 kvůli lepší kompatibilitě se staršími aplikacemi. Dokumenty je možné ukládat i v jiných kódováních, ale pak musíme toto kódování povinně určit v deklaraci XML a nemáme jistotu, že tento dokument zvládnou zpracovat všechny aplikace.

V případě, že potřebujeme do dokumentu vložit nějaký znak, který buď není snadno dostupný na klávesnici nebo nejde reprezentovat v použitém kódování, můžeme do dokumentu vložit odkaz na číselný kód znaku v Unicode (podrobnější vysvětlení problematiky Unicode a kódování naleznete v kapitole 3). Předchozí dokument XML tak můžeme také zapsat jako:

```
<?xml version="1.0" encoding="us-ascii"?>
<osoba>
  <jméno>Jan</jméno>
  <příjmení>Nov&#xE1;k</příjmení>
  <věk>42</věk>
</osoba>
```

Znak „á“ byl nahrazen svým číselným kódem (U+00E1) zapsaným v šestnáctkové soustavě. Kód je možné zapsat i v desítkové soustavě, v číselném odkazu na znak pak chybí ‚x‘.

```
<?xml version="1.0" encoding="us-ascii"?>
<osoba>
  <jméno>Jan</jméno>
  <příjmení>Nov&#225;k</příjmení>
  <věk>42</věk>
</osoba>
```

# 1.7 Komentáře

Pokud potřebujeme v dokumentu něco vysvětlit nebo část textu dočasně skrýt, s výhodou k tomu použijeme komentář. Komentář je součástí dokumentu, ale parsery jej ignorují a není dále zpracováván. Komentář se zapisuje mezi znaky <!-- a -->.

```
<!-- Vysvětlující text -->
```

Komentář může obsahovat cokoliv, kromě posloupnosti znaků --. Z toho vyplývá, že komentáře do sebe bohužel nemůžeme zanořovat. V komentáři dokonce můžeme používat tagy atd. Jsou však zcela ignorovány. To se hodí pro dočasné vyřazení části dokumentu ze zpracování.

```
<para>První odstavec.</para>
<!-- <para>Šéf mi leze krkem.</para> -->
<para>Třetí odstavec.</para>
```



# 1.8 Sekce CDATA

Pokud potřebujeme do dokumentu vložit větší kus textu, kde se hojně používají znaky se speciálním významem jako ‚<', ‚>' a ‚&', je nepohodlné je zapisovat pomocí znakových entit. V takových případech se používá tzv. sekce CDATA. Oceníme ji zejména v případech, kdy je součástí XML dokumentu kód nějakého programu nebo HTML či XML kód. Použití sekcí CDATA si ukážeme na následujícím dokumentu.

```
<script type="text/javascript"><![CDATA[
  for (i=0; i < 10; i++)
  {
    document.writeln("<p>Ahoj</p>");
  }]]>
</script>
```

Bez použití sekce CDATA by byl zápis přece jen poněkud krkolomný.

```
<script type="text/javascript">
  for (i=0; i &lt; 10; i++)
  {
    document.writeln("&lt;p&gt;Ahoj&lt;/p&gt;");
  }
</script>
```

Obecně se tedy sekce CDATA zapisují jako <![CDATA[*«text»*]]>. Text přitom může obsahovat cokoliv, kromě sekvence znaků ]]>. Konstrukce CDATA existuje v XML pouze pro větší pohodlí autorů, kteří zapisují XML kód ručně. Nepřidává do XML žádnou novou sémantiku, jde jen o alternativní syntaxi.

# 1.9 Instrukce pro zpracování

XML dokumenty mohou být zpracovávány různými programy. Někdy může být užitečné do dokumentu uložit řídící informace, které jsou určeny pouze pro některý program. Můžeme tak do dokumentu zařadit odkaz na styl definující zobrazení v prohlížeči, formátovacímu programu můžeme naznačit, kde má zalomit stránku. Moderní skriptové jazyky pro generování dynamických webových stránek se také zapisují přímo do těla dokumentů. Pro všechny tyto účely má XML k dispozici standardní způsob pro zařazení nestandardních informací. Na libovolné místo dokumentu (kromě značkovaní – podobně jako u komentářů) můžeme vložit *instrukce pro zpracování (processing instructions)*. Tyto instrukce XML parser ignoruje, předá je nadřazené aplikaci – záleží na ní, zda je nějak využije. Syntaxe instrukcí je velice jednoduchá.

<?*«identifikátor»* *«data»*?>

Pomocí *«identifikátoru»* můžeme rozlišovat jednotlivé druhy instrukcí – do jednoho dokumentu můžeme umístit instrukce pro několik různých programů. Samotná *«data»* instrukce mohou mít libovolný tvar, ale nesmějí obsahovat sekvenci znaků ?>.

Pomocí instrukcí pro zpracování lze do dokumentů zařadit například příkazy skriptovacího jazyka PHP.

```
<dokument>
  <datum>Dnešní datum je <?php echo Date("d.m.Y")?></datum>
  <para>Nějaké důležité informace.</para>
</dokument>
```

Pomocí instrukcí pro zpracování se k XML dokumentu připojují i styly definující zobrazení v prohlížeči.

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet href="styl.css" type="text/css"?>
<clanek>
  <zahlavi>
    <rubrika>Téma týdne</rubrika>
    <nazev>XML a stylové jazyky</nazev>
    <autor>Jiří Kosek</autor>
  </zahlavi>
  ...
</clanek>
```

# 1.10 Automatická kontrola syntaxe

Splňuje-li dokument všechna výše uvedená pravidla, je syntakticky v pořádku a říkáme o něm, že je *správně strukturovaný (well-formed)*.

Správnou syntaxi si můžeme nechat zkontrolovat pomocí tzv. *parseru*. Jednoduchý parser XML je dnes obsažen v každém webovém prohlížeči, stačí v něm otevřít dokument XML a v případě chyby dostaneme chybové hlášení. Můžeme si to vyzkoušet na následujícím dokumentu, který obsahuje dvě chyby – překlep v názvu koncového tagu `jméno` a neukončený element `věk`.

```
<?xml version="1.0" encoding="utf-8"?>
<osoba>
  <jméno>Jan</jémno>
  <příjmení>Novák</příjmení>
  <věk>42
</osoba>
```

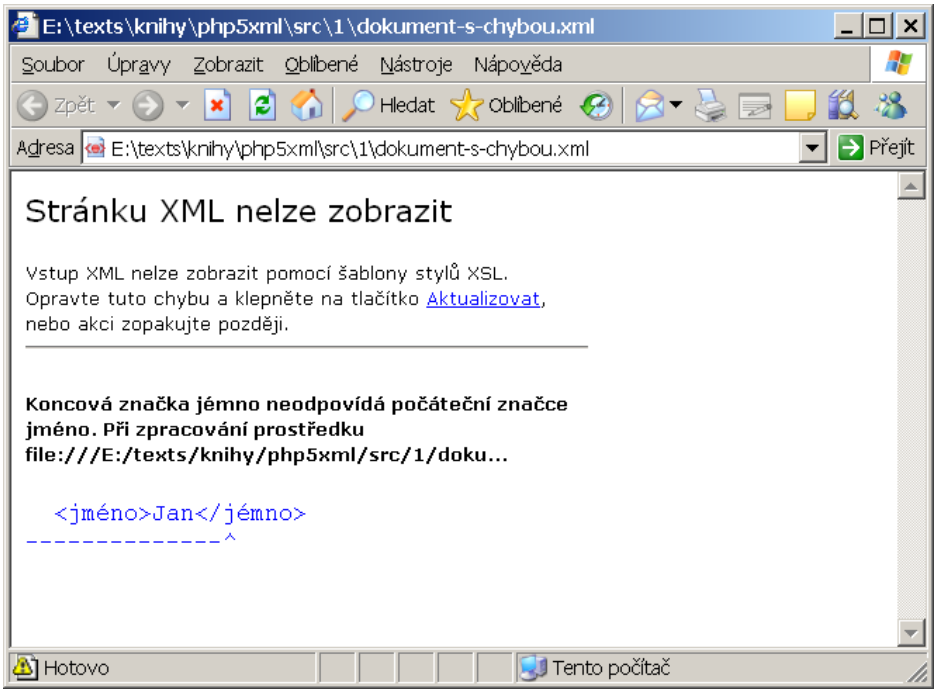
Prohlížeče se většinou zastaví na první chybě, kterou naleznou (viz obrázek 1.2). Mnoho parserů je dostupných i v podobě jednoduchého programu, který můžeme spouštět z příkazové řádky. Velice rychlý a na funkce bohatý je parser **xmllint**[2], který pro práci s XML používá stejnou knihovnu libxml2 jako PHP. Následující ukázka zobrazuje výstup programu **xmllint** při zpracování dokumentu z předešlého příkladu.

```
$ xmllint --noout dokument-s-chybou.xml
dokument-s-chybou.xml:3: parser error : Opening and ending tag mismatch: jméno ▶
line 3 and jémno
  <jméno>Jan</jémno>
                   ^
dokument-s-chybou.xml:6: parser error : Opening and ending tag mismatch: věk ▶
line 5 and osoba
</osoba>
        ^
dokument-s-chybou.xml:6: parser error : Premature end of data in tag osoba line 2
</osoba>
        ^
```

# 1.11 Jmenné prostory

Jedním ze základních cílů jazyka XML je poskytnout aplikacím formát, ve kterém půjde vyměnovat informace po celém světě. Pro dosažení tohoto úkolu je však potřeba zajistit, aby byly elementy používané v dokumentech jednoznačně identifikované a navzájem rozlišitelné. Jinak nebudeme například schopni rozlišit, zda element `název` popisuje název knihy v katalogu knihkupectví, nebo obchodní název firmy ve výpisu z obchodního rejstříku, nebo ještě něco úplně jiného. Nutnost jednoznačného rozlišení elementů je důležitá v těch případech, kdy přesně nevíme, jaké informace zpracovávaný dokument obsahuje, nebo zpracováváme komponovaný dokument, který obsahuje elementy z několika různých oblastí.

[2] *http://xmlsoft.org/*




*Obrázek 1.2: Zobrazení chyby v dokumentu XML*

Problém jednoznačné identifikace elementů v dokumentech XML řeší *jmenné prostory*. Používáme-li v dokumentu XML jmenné prostory, není už element jednoznačně identifikován jen svým jménem, ale kombinací jména a jmenného prostoru. Jmenný prostor má přitom podobu adresy URI, která zajišťuje možnost celosvětově vytvářet nová URI a přitom zachovat jejich unikátnost. Důležité je uvědomit si, že URI adresa v tomto případě slouží jen jako identifikátor, aplikace pracující s XML se nikdy nesnaží z této adresy získat nějaký dokument.

Pro rozlišení dvou dříve zmíněných významů elementu `název` tak můžeme použít dva různé jmenné prostory. V následující fiktivní syntaxi doplníme před názvy elementů URI jmenného prostoru.

```
<{http://knihkupectvi.cz/katalog}název>Čuk a Gek</{http://knihkupectvi.cz/katalog}název>

<{http://justice.cz/ns/or}název>Grada</{http://justice.cz/ns/or}název>
```

Vidíme, že kombinace URI jmenného prostoru a název elementu je teď už jednoznačná a je možné odlišit, kdy se o jaký název jedná. Zároveň však vidíme, že výše uvedený zápis by byl velmi nepohodlný a zdlouhavý. Ve skutečnosti se ani nejedná o syntaxi, která by fungovala, šlo mi jen o naznačení principu. V dokumentech se pro usnadnění zápisu příslušnosti elementu do nějakého jmenného prostoru používá jedna z následujících dvou syntaxí.

První možností je použití *implicitního (výchozího) jmenného prostoru*. Chceme-li, aby nějaký element a všichni jeho potomci (tj. elementy v něm obsažené) patřily do nějakého jmenného prostoru, stačí když u tohoto elementu nadeklarujeme požadovaný jmenný prostor jako implicitní pomocí atributu `xmlns`. Následující ukázka je dokument v jazyce

XHTML, kde všechny elementy patří do jmenného prostoru `http://www.w3.org/1999/xhtml`.

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Ukázka XHTML stránky</title>
  </head>
  <body>
    <h1>Ukázka XHTML stránky</h1>
    <p>Všechny elementy v tomto dokumentu patří
       do jmenného prostoru XHTML.</p>
  </body>
</html>
```

Druhá varianta spočívá v deklaraci prefixu, který zastupuje zvolený jmenný prostor. Tento prefix se pak zapisuje před jména všech elementů patřících do jmenného prostoru. Deklarace prefixu jmenného prostoru se provádí pomocí atributu ve speciálním tvaru:

```
xmlns:«prefix»="«URI»"
```

Náš ukázkový XHTML dokument proto můžeme zapsat také následujícím způsobem:

```
<?xml version="1.0" encoding="UTF-8"?>
<html:html xmlns:html="http://www.w3.org/1999/xhtml">
  <html:head>
    <html:title>Ukázka XHTML stránky</html:title>
  </html:head>
  <html:body>
    <html:h1>Ukázka XHTML stránky</html:h1>
    <html:p>Všechny elementy v tomto dokumentu patří
      do jmenného prostoru XHTML.</html:p>
  </html:body>
</html:html>
```

Zápisu jména elementu ve tvaru `html:title` se říká *kvalifikované jméno elementu (QName)*. To se skládá z prefixu (`html`) a z lokálního jména (`title`). Kombinace lokálního jména a jmenného prostoru, pro který je prefix deklarován, společně jednoznačně identifikuje element. Prefix ovšem může být libovolný, slouží jen jako pomůcka pro zkrácení zápisu. Vždy je však důležité, jaký jmenný prostor zastupuje.

Jmenné prostory nacházejí uplatnění zejména v dokumentech, které se skládají z různých sad značek. V praxi je takových případů hodně. Můžeme mít například XHTML dokument, který obsahuje vložené obrázky ve formátu SVG a matematické vzorce v MathML. Elementy těchto tří značkovacích jazyků jsou přitom ve zvláštních jmenných prostorech, abychom je dokázali rozlišit. Styly zapsané v jazyce XSLT zase používají jmenné prostory k odlišení výkonných instrukcí XSLT od elementů, které se jen kopírují na výstup transformace.

Následující ukázka zachycuje dokument v XHTML, který obsahuje vložený fragment kódu s obrázkem v SVG.

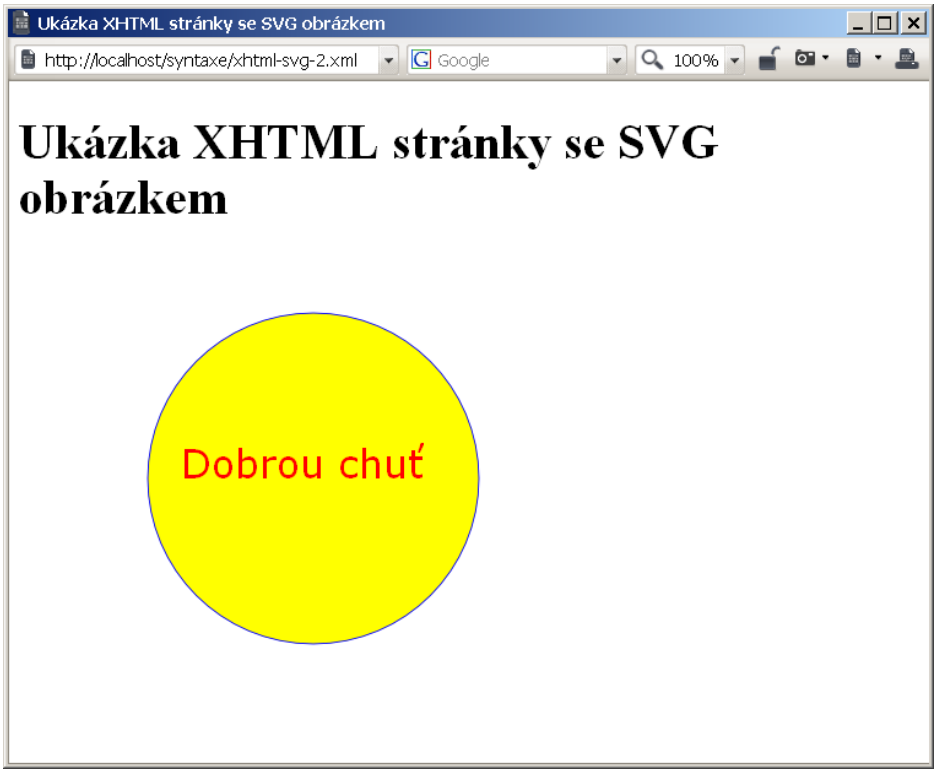
```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
```

```
  <head>
    <title>Ukázka XHTML stránky se SVG obrázkem</title>
  </head>
  <body>
    <h1>Ukázka XHTML stránky se SVG obrázkem</h1>
    <svg:svg width="4in" height="3in" viewBox="0 0 400 400">
      <svg:title>Žlutý kruh s červeným nápisem</svg:title>
      <svg:g>
        <svg:circle style="fill: yellow; stroke: blue"
                    cx="200" cy="200" r="150"/>
        <svg:text x="80" y="200"
          style="font-size: 36px; font-family: Verdana;
                 color: red; fill: red">Dobrou chuť</svg:text>
      </svg:g>
    </svg:svg>
  </body>
</html>
```

Deklarace prefixu jmenného prostoru nebo implicitního jmenného prostoru platí pro element, ve kterém je uvedena, a pro všechny jeho podelementy. Pokud však na některém z podelementů prefix nebo implicitní jmenný prostor předefinujeme, platí nová definice. Předchozí dokument tak můžeme zapsat i následujícím způsobem:

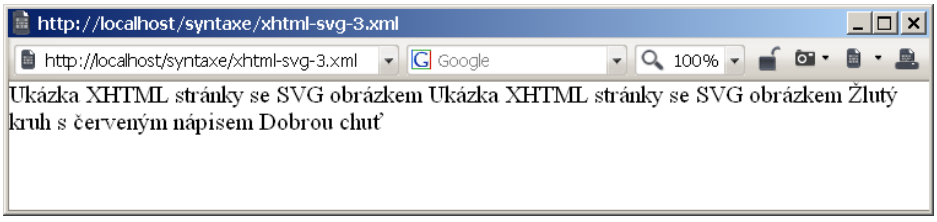
```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Ukázka XHTML stránky se SVG obrázkem</title>
  </head>
  <body>
    <h1>Ukázka XHTML stránky se SVG obrázkem</h1>
    <svg xmlns="http://www.w3.org/2000/svg"
         width="4in" height="3in" viewBox="0 0 400 400">
      <title>Žlutý kruh s červeným nápisem</title>
      <g>
        <circle style="fill: yellow; stroke: blue" cx="200" cy="200" r="150"/>
        <text x="80" y="200"
          style="font-size: 36px; font-family: Verdana;
                 color: red; fill: red">Dobrou chuť</text>
      </g>
    </svg>
  </body>
</html>
```

V takto složených dokumentech přitom jmenné prostory neslouží jen k odlišení jednotlivých elementů, ale především k jejich jednoznačné identifikaci. Ví-li prohlížeč, že nějaké elementy patří do jmenného prostoru XHTML, a jiné zase do jmenného prostoru SVG, může je podle toho interpretovat. Ukazuje to obrázek 1.3, kde je přímo ve stránce vložený SVG obrázek vykreslen.




*Obrázek 1.3: Zobrazení XHTML stránky s vloženým obrázkem SVG*

Obrázek 1.4 zachycuje případ, kdy jsme elementy SVG neumístili do jmenného prostoru, prohlížeč je tedy nerozpoznal a nezpracoval jako SVG obrázek.




*Obrázek 1.4: Zobrazení XHTML stránky se špatně vloženým SVG*

Atributy se v běžných případech do jmenného prostoru neumisťují, a proto se na ně ani nevztahuje implicitní jmenný prostor. Chápe se to tak, že atribut patří vždy k elementu, u kterého je uveden, a tento element už do nějakého jmenného prostoru patří.

Nicméně i atributy patřící do nějakého jmenného prostoru mají své uplatnění. Říká se jim globální atributy a jedná se o obecné atributy, které lze použít u jakýchkoliv elementů a přidávají jim speciální sémantiku. Např. v jmenném prostoru `http://www.w3.org/2001/XMLSchema-instance` jsou k dispozici atributy určující umístění schématu nebo neurčenou hodnotu.

```
<osoba xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="osoba.xsd">
  <jméno>Jenda</jméno>
  <narozen xsi:nil="true"></narozen>
</osoba>
```

Atributy `noNamespaceSchemaLocation` a `nil` jde použít u libovolného elementu a právě proto, aby nemohlo dojít ke kolizi s jinými atributy, jsou umístěny ve speciálním jmenném prostoru.

Podobně fungují atributy standardu XLink, které umožňují z libovolného elementu udělat odkaz. Například:

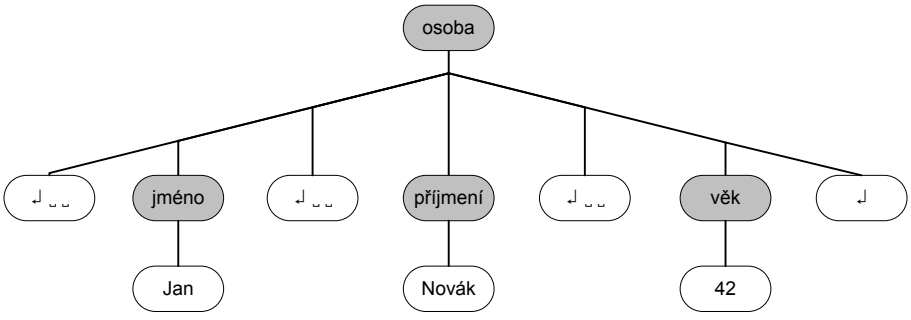
```
<para xmlns:xlink="http://www.w3.org/1999/xlink">Pro zjištění
délky řetězce můžeme použít funkci
<function xlink:href="http://php.net/strlen">strlen</function>.
</para>
```

# 1.12 Práce s bílými znaky

V dokumentu XML se obvykle vyskytuje velké množství bílých znaků (mezer, konců řádek, tabulátorů), které slouží pro zpřehlednění zápisu – obvykle se každý element uvádí na novém řádku a velikost jeho odsazení odpovídá hloubce jeho zanoření do ostatních elementů. Tyto bílé znaky však obecně nejde pokládat za zbytečné, a proto jsou vždy parserem předány aplikaci k dalšímu zpracování. Bílé znaky použité pro zpřehlednění zápisu se v datovém modelu dokumentu XML většinou projeví jako textové uzly, které se skládají pouze z těchto bílých znaků.

Popsané chování demonstruje i následující ukázka. Strom dokumentu XML obsahuje mezi všemi elementy textové uzly s bílými znaky. Je důležité si uvědomit, že tyto uzly jsou součástí dokumentu a při zpracování s nimi počítat.

```
<osoba>
  <jméno>Jan</jméno>
  <příjmení>Novák</příjmení>
  <věk>42</věk>
</osoba>
```



V našem příkladě je evidentní, že textové uzly s bílými znaky jsou skutečně pro obsah dokumentu zcela zbytečné a bylo by možné je ignorovat. Knihovna libxml2 dokonce umožňuje tyto uzly automaticky odstranit. Jak by dokument dopadl po odstranění uzlů s bílými znaky, můžeme zkontrolovat i pomocí řádkové utility **xmllint**.

```
$ xmllint --noblanks osoba.xml
<?xml version="1.0" encoding="utf-8"?>
<osoba><jméno>Jan</jméno><příjmení>Novák</příjmení><věk>42</věk></osoba>
```

V praxi se však setkáme i s případy, kdy je odstranění textových uzlů s bílými znaky nežádoucí. Jedná se o tzv. *smíšený obsah*. Elementy se smíšeným obsahem jsou takové elementy, které mohou obsahovat jak přímo text, tak další podelementy. Typickým příkladem smíšeného obsahu jsou odstavce – ty obsahují text, ale v něm se mohou vyskytovat další elementy například pro zvýraznění textu nebo pro vytváření odkazů. Textové uzly s bílými znaky ve smíšeném obsahu často nesou důležitou informaci, jako mezery mezi slovy. Vše ukazuje následující dokument.

```
<?xml version="1.0" encoding="utf-8"?>
<dokument>
  <p>První odstavec obsahuje na první pohled smíšený obsah.
     Tady se nám <em>Jan</em> <i>Novák</i> neslije dohromady.</p>
  <p><em>Jan</em> <i>Novák</i></p>
</dokument>
```

Kdybychom teď nechali parser odstraňovat uzly s bílými znaky, chybně v druhém odstavci odstraní mezeru mezi slovy Jan a Novák, a dostaneme tak nesmyslný text.

```
$ xmllint --noblanks odstavec.xml
<?xml version="1.0" encoding="utf-8"?>
<dokument><p>První odstavec obsahuje na první pohled smíšený obsah.
     Tady se nám <em>Jan</em> <i>Novák</i> neslije dohromady.</p><p><em>Jan</▶
em><i>Novák</i></p></dokument>
```

Proč se tak stalo? Důvod je jednoduchý, parser se bez nějakých přídavných znalostí nemůže správně rozhodnout, kdy se jedná o smíšený obsah a kdy ne. U prvního odstavce šlo jednoznačně o smíšený obsah, protože se na stejné úrovni vyskytoval text i elementy. V druhém odstavci se však uvnitř elementu `p` objevily jen bílé znaky a další elementy. Nešlo jej tedy rozlišit od situace, kdy jsou uzly s bílými znaky v dokumentu čistě pro okrasu.

Vidíme tedy, že říkat parseru o odstranění bílých znaků je bezpečné pouze v případě, kdy jsme si jistí, že v dokumentu nepoužíváme smíšený obsah. Pokud to neuděláme, hrozí, že při zpracování dokumentu přijdeme o některé podstatné informace.

Aby mohl parser správně poznat, které elementy obsahují smíšený obsah a které ne, muselo by k němu být připojené schéma dokumentu. Z něj lze poznat, které elementy obsahují jen další podelementy a které mají smíšený obsah. Kdybychom k dokumentu doplnili například DTD (více si o nich povíme v části 9.2), odstranění bílých znaků proběhne bezpečně.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE dokument [
  <!ELEMENT dokument    (p+)>
  <!ELEMENT p           (#PCDATA|em|i)*>
  <!ELEMENT em          (#PCDATA)>
  <!ELEMENT i           (#PCDATA)>
```

```
]>
<dokument>
  <p>První odstavec obsahuje na první pohled smíšený obsah.
     Tady se nám <em>Jan</em> <i>Novák</i> neslije dohromady.</p>
  <p><em>Jan</em> <i>Novák</i></p>
</dokument>

$ xmllint --noblanks odstavec-dtd.xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE dokument [
<!ELEMENT dokument (p)+>
<!ELEMENT p (#PCDATA | em | i)*>
<!ELEMENT em (#PCDATA)>
<!ELEMENT i (#PCDATA)>
]>
<dokument><p>První odstavec obsahuje na první pohled smíšený obsah.
     Tady se nám <em>Jan</em> <i>Novák</i> neslije dohromady.</p><p><em>Jan</▶
em> <i>Novák</i></p></dokument>
```

Vidíme, že uzly s bílými znaky se odstranily jen mezi elementy `dokument` a `p` tak, jak to určuje DTD. Element `p` je definován jako smíšený obsah, a proto se v něm bílé znaky neodstraňují.

Bílým znakům jsme se věnovali poněkud více, protože je to téma, ve kterém má mnoho lidí poměrně dost nejasností. Je to umocněno i tím, že výchozí konfigurace parseru MSXML od Microsoftu se chová nestandardně a všechny uzly s bílými znaky vypouští.

Každý parser navíc provádí určité úpravy bílých znaků, aby usnadnil práci aplikaci, která dokumenty XML zpracovává (v našem případě tedy PHP skriptu). První úprava spočívá v normalizaci znaků pro konec řádku. Různé počítačové platformy používají různé znaky pro konec řádku – ve Windows je to sekvence znaků CR LF, na unixech znak LF a na Macovi CR. Parser XML proto všechny tyto kombinace vždy převede na znak LF (kód tohoto znaku je 10).

O něco komplexnější je normalizace hodnot atributů. Nejprve jsou v hodnotě atributu znormalizovány znaky konce řádku na LF a poté se všechny bílé znaky (tedy LF, mezery a tabulátory) převedou na mezery.[3]

# 1.13 Skládání dokumentů

Jsou situace, kdy se kousek textu nebo značkování v dokumentu opakuje na několika místech a my ho nechceme opisovat pořád dokola. Jindy zase chceme určitou část kódu XML používat ve více různých dokumentech XML najednou. Pro oba tyto úkoly nabízí XML nástroje – jednak starší založené na entitách a novější založené na standardu XInclude.

[3] Je-li navíc atribut deklarován v DTD a má jiný typ než CDATA, je normalizace ještě složitější. V případě potřeby se na její popis můžete podívat do specifikace XML *http://www.w3.org/TR/REC-xml/#AVNormalize*.

## 1.13.1 Entity

Každý dokument XML se může skládat z několika entit. Všechny dokumenty, které jsme zatím viděli, se skládaly pouze z jedné entity, která tvořila celý dokument. Na začátku dokumentu v *deklaraci typu dokumentu* můžeme definovat jednu nebo více entit, na které se pak můžeme dále v těle dokumentu odkazovat. Každý odkaz na entitu se nahradí fragmentem kódu XML, který zastupuje.

Deklarace entit se nejčastěji uvádějí v tzv. *interní podmnožině*:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE «kořenový element» [
  «deklarace entit»
]>
…
```

Entity je možné definovat i v *externí podmnožině*, která obvykle obsahuje schéma dokumentu v podobě DTD a používá se ve více dokumentech. Podrobnější výklad tohoto způsobu najdete v 9.2.

Deklarace entity má přitom tvar:

```
<!ENTITY «název entity» …>
```

Takto deklarovanou entitu pak můžeme použít v dokumentu pomocí *odkazu na entitu*, který má tvar:

```
&«název entity»;
```

### 1.13.1.1 Interní textové entity

Interní textové entity umožňují deklarovat entitu, která zastupuje často používaný text, kus XML kódu nebo třeba jen znak těžko dostupný na klávesnici. Odkaz na tuto entitu pak můžeme podle libosti používat dále v dokumentu v obsahu elementů nebo atributů.

***Příklad 1.1:*** *Ukázka interních textových entit – `syntaxe/interni-entity.xml`*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE manuál [
<!ENTITY program "Headache 2.7">
<!ENTITY nbsp    " ">
]>
<manuál>
  <název>&program; - Uživatelská příručka</název>
  <odstavec>Program &program; můžete spustit výběrem příkazu v menu.</▶
odstavec>
  <odstavec>Odinstalování aplikace &program; není možné.</odstavec>
</manuál>
```

Při zpracování dokumentu se odkazy na entity automaticky nahradí textem, který entita zastupuje. Ověřit si to můžeme zase pomocí programu **xmllint** a jeho volby `--noent`.

```
$ xmllint --noent interni-entity.xml
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE manuál [
<!ENTITY program "Headache 2.7">
<!ENTITY nbsp " ">
]>
<manuál>
  <název>Headache 2.7 - Uživatelská příručka</název>
  <odstavec>Program Headache 2.7 můžete spustit výběrem příkazu v menu.</▶
odstavec>
  <odstavec>Odinstalování aplikace Headache 2.7 není možné.</odstavec>
</manuál>
```

### 1.13.1.2 Externí textové entity

Externí textové entity umožňují dokument složit dohromady z několika samostatných souborů. Je to užitečné zejména tehdy, kdy potřebujeme ručně zpracovávat dlouhý dokument. Rozdělením dokumentu do několika souborů získáme kratší dokumenty, které se snadněji editují. Problém externích entit je v tom, že samostatné entity nemohou obsahovat vlastní deklaraci typu dokumentu, a tudíž mohou používat pouze entity definované v „hlavním" dokumentu a nelze je zpracovat samostatně (používají-li entity).

***Příklad 1.2:*** *Externí entita – `syntaxe/kapitola.xml`*

```
<?xml version="1.0" encoding="UTF-8"?>
<kapitola>
  <název>Úvod</název>
  <odstavec>Tak tohle je text samostatné kapitoly.</odstavec>
  <odstavec>Trošku ten text ještě prodloužíme.</odstavec>
</kapitola>
```

***Příklad 1.3:*** *Soubor načítající externí entitu – `syntaxe/externi-entity.xml`*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE manuál [
<!ENTITY uvod SYSTEM "kapitola.xml">
]>
<manuál>
  <název>Uživatelská příručka</název>
  &uvod;
</manuál>
```

V místě odkazu na entitu `&uvod;` se do dokumentu vloží celý obsah externího souboru `kapitola.xml`. Můžeme se o tom přesvědčit.

```
$ xmllint --noent externi-entity.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE manuál [
<!ENTITY uvod SYSTEM "kapitola.xml">
]>
<manuál>
  <název>Uživatelská příručka</název>

<kapitola>
  <název>Úvod</název>
  <odstavec>Tak tohle je text samostatné kapitoly.</odstavec>
```

```
  <odstavec>Trošku ten text ještě prodloužíme.</odstavec>
</kapitola>
</manuál>
```

V deklaraci entity za klíčovým slovem `SYSTEM` můžeme použít jakoukoliv adresu URI, nejčastěji se proto používá adresa URL, ať už relativní nebo absolutní.

## 1.13.2 XInclude

Skládání dokumentů pomocí externích entit má některé nevýhody – externí entity nemohou mít vlastní deklaraci typu dokumentu a navíc je mechanismus entit nepřímý – nejdříve entitu deklarujeme a teprve poté ji používáme. Navíc syntaxe pro deklaraci entit poněkud vybočuje ze syntaxe pro zápis elementů a atributů, protože byla do XML převzata z jeho historického předchůdce jazyka SGML.

Všechna tato omezení řeší standard XInclude. Ten definuje element `include` ve jmenném prostoru `http://www.w3.org/2001/XInclude`. Sémantika tohoto elementu je taková, že element se nahradí souborem, na který ukazuje jeho atribut `href`.

***Příklad 1.4:*** *Složení dokumentu pomocí XInclude – `syntaxe/xinclude.xml`*

```
<?xml version="1.0" encoding="UTF-8"?>
<manuál>
  <název>Uživatelská příručka</název>
  <xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
              href="kapitola.xml"/>
</manuál>
```

Pro otestování skládání entit můžeme opět využít **xmllint**, tentokrát mu však pomocí parametru `--xinclude` řekneme, aby vyhodnotil elementy XInclude, na které v dokumentu narazí.

```
$ xmllint --xinclude xinclude.xml
<?xml version="1.0" encoding="UTF-8"?>
<manuál>
  <název>Uživatelská příručka</název>
  <kapitola>
  <název>Úvod</název>
  <odstavec>Tak tohle je text samostatné kapitoly.</odstavec>
  <odstavec>Trošku ten text ještě prodloužíme.</odstavec>
</kapitola>
</manuál>
```

XInclude umožňuje nahradit i interní entity, byť už ne tolik elegantním způsobem. Definice všech textů, které se mají opakovaně používat, můžeme umístit do samostatného souboru a každému elementu přiřadit unikátní identifikátor, jak ukazuje následující příklad.

***Příklad 1.5:*** *Definice sdílených elementů – `syntaxe/definice.xml`*

```
<?xml version="1.0" encoding="UTF-8"?>
<definice>
  <text xml:id="program">Headache 2.7</text>
</definice>
```

V adrese elementu XInclude můžeme za znakem ‚#' uvést identifikátor elementu nebo i složitější XPointer výraz, které určí, jaká část z celého odkazovaného elementu se má do dokumentu vložit.

***Příklad 1.6:*** *Nahrazení interních entit pomocí XInclude – `syntaxe/xinclude-fragmenty.xml`*

```
<?xml version="1.0" encoding="UTF-8"?>
<dokument xmlns:xi="http://www.w3.org/2001/XInclude">
  <název>Uživatelská příručka</název>
  <!-- načtení celého elementu určeného pomocí ID -->
  <xi:include href="definice.xml#program"/>
  <!-- načtení textu elementu -->
  <xi:include href="definice.xml#xpointer(id('program')/text())"/>
</dokument>
```

Po složení pak bude dokument vypadat následovně:

```
$ xmllint --xinclude xinclude-fragmenty.xml
<?xml version="1.0" encoding="UTF-8"?>
<dokument xmlns:xi="http://www.w3.org/2001/XInclude">
  <název>Uživatelská příručka</název>
  <!-- načtení celého elementu určeného pomocí ID -->
  <text xml:id="program">Headache 2.7</text>
  <!-- načtení textu elementu -->
  Headache 2.7
</dokument>
```

XInclude můžeme použít i pro vložení textového souboru, který se nemá chápat jako dokument XML obsahující značkování. Pomocí přídavných atributů určíme, že se jedná o textový soubor a v jakém je uložen kódování.

***Příklad 1.7:*** *Vložení textového souboru pomocí XInclude*

```
<?xml version="1.0" encoding="UTF-8"?>
<dokument xmlns:xi="http://www.w3.org/2001/XInclude">
  <název>Výpis PHP skriptu</název>
  <xi:include href="demo.php" parse="text" encoding="iso-8859-2"/>
</dokument>
```

```
$ xmllint --xinclude xinclude-text.xml
<?xml version="1.0" encoding="UTF-8"?>
<dokument xmlns:xi="http://www.w3.org/2001/XInclude">
  <název>Výpis PHP skriptu</název>
  &lt;?php
  // ukázkový skript v PHP
  echo "Ahoj";
  phpinfo();
?&gt;
</dokument>
```

# 1.14 Katalogové soubory

Mnoho dokumentů XML nejde zpracovat samostatně, ale pro jejich korektní zpracování je potřeba načíst i další soubory s přídavnými informacemi. Vezměme si jako příklad stránku v jazyce XHTML. Začátek takové stránky vypadá následovně:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html lang="cs" xml:lang="cs" xmlns="http://www.w3.org/1999/xhtml">
```

Vidíme, že druhá řádka obsahuje deklaraci typu dokumentu, která ukazuje na DTD definující elementy a atributy přípustné uvnitř dokumentů XHTML. DTD je přitom umístěno na webovém serveru, takže například při pokusu o validaci dokumentu se musí celé DTD nejprve stáhnout a teprve poté se provede samotná validace.

```
$ xmllint --noout --valid stranka.xhtml
```

I s rychlým připojením k internetu bude validace trvat pár sekund, kdy čekáme, než se stáhne DTD. Co se stane, když nebudeme připojeni k síti? Parseru se nepodaří získat DTD a pokus o načtení dokumentu selže.

```
$ xmllint --noout --valid stranka.xhtml
stranka.xhtml:3: warning: failed to load external entity "http://www.w3.org/TR/▶
xhtml1/DTD/xhtml1-strict.dtd"
                      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
                                                                         ^
stranka.xhtml:4: validity error : Validation failed: no DTD found !
<html lang="cs" xml:lang="cs" xmlns="http://www.w3.org/1999/xhtml">
                                                                ^
stranka.xhtml:6: parser error : Entity 'ndash' not defined
    <title>Slohová práce &ndash; Jan Novák</title>
                                ^
stranka.xhtml:12: parser error : Entity 'nbsp' not defined
    <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
                        ^
```

Vidíme, že snaha o validaci dokumentu, který DTD načítá ze sítě, nemusí být vždy úspěšná, a i když je úspěšná, tak je poměrně pomalá. Jak tento stav vylepšit? Můžeme si samozřejmě říci, že přece nepotřebujeme validaci provádět a nemusíme tedy načítat DTD. To bohužel není pravda, protože DTD kromě deklarací elementů a atributů obsahuje i deklarace entit, které v dokumentu běžně používáme (např. v XHTML jsou to entity jako ` `, `&copy;` a `&ndash;`). Takže i když validaci oželíme, nepůjde dokument bez DTD korektně zpracovat, protože parser nebude vědět, jakým textem má nahradit odkazy na entity.

Řešením tohoto problému je umístit si kopii DTD na svém lokálním počítači a přinutit parser, aby tuto lokální kopii používal místo DTD umístěného někde na internetu. Většina parserů je schopná řídit se při zpracování dokumentů *katalogovým souborem*, který slouží k mapování odkazů na externí entity (včetně DTD) na jejich lokální kopie.

V našem případě tedy stačí stáhnout si DTD pro XHTML[4] a uložit je do nějakého adresáře. V dalším textu budeme předpokládat, že jsme DTD uložili do adresáře `c:\data\xhtml` (resp. `/data/xhtml` na unixovém systému). V adresáři `data` si nyní vytvoříme katalogový soubor.

***Příklad 1.8:*** *Ukázka katalogového souboru – `data/catalog.xml`*

```
<?xml version='1.0' encoding="utf-8"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">

<public publicId="-//W3C//DTD XHTML 1.0 Transitional//EN"
        uri="xhtml/xhtml1-transitional.dtd"/>

<public publicId="-//W3C//DTD XHTML 1.0 Strict//EN"
        uri="xhtml/xhtml1-strict.dtd"/>

</catalog>
```

Vidíme, že katalogový soubor mapuje veřejné identifikátory XHTML `-//W3C//DTD XHTML 1.0 Transitional//EN` a `-//W3C//DTD XHTML 1.0 Strict//EN` na lokální kopie DTD v podadresáři `xhtml`.

Parseru nyní stačí říci, aby tento katalog používal. Program **xmllint** podporuje několik způsobů, jak určit umístění katalogového souboru. Prvním z nich je proměnná prostředí `XML_CATALOG_FILES`. Očekává se, že bude obsahovat absolutní URI ke katalogovému souboru. Tj.

```
file:///c:/data/catalog.xml
```

resp.

```
file:///data/catalog.xml
```

Způsob nastavení této proměnné záleží na použitém operačním systému. Můžeme využít například příkaz **set**.

```
c:\data>set XML_CATALOG_FILES=file:///c:/data/catalog.xml
c:\data>xmllint --noout --valid stranka.xhtml
```

Vidíme, že validace teď funguje i bez připojení k síti a je velmi rychlá, protože se nečtou žádné další soubory ze sítě.

Kromě využití proměnné `XML_CATALOG_FILES` můžeme na unixových systémech katalog uložit do souboru `/etc/catalog`, kde se standardně očekává.

Neobsahuje-li deklarace typu dokumentu veřejný identifikátor, např.:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html SYSTEM "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html lang="cs" xml:lang="cs" xmlns="http://www.w3.org/1999/xhtml">
```

Můžeme do katalogového souboru přidat i položky, které zajistí přesměrování pro systémové identifikátory:

[4] *http://www.w3.org/TR/xhtml1/xhtml1.zip*

```
<system systemId="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
        uri="xhtml/xhtml1-strict.dtd"/>

<system systemId="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
        uri="xhtml/xhtml1-transitional.dtd"/>
```

Takto nastavené katalogové soubory umí využívat všechny XML funkce obsažené v PHP5. Používají se pro přesměrování načítání nejen pro DTD, ale i pro importované styly XSLT apod.

# 1.15 Speciální atributy

Jazyk XML a na něj navazující standardy definuje několik univerzálních atributů, které lze použít na libovolném elementu. Všechny tyto atributy jsou globální a patří do jmenného prostoru `http://www.w3.org/XML/1998/namespace`. Tento jmenný prostor je speciální a nemusí se pro něj deklarovat prefix. Existuje pro něj předdefinovaný prefix `xml`.



## 1.15.1 xml:lang

Pomocí atributu `xml:lang` můžeme pro element určit jazyk, v jakém je zapsán jeho obsah. Tuto informaci pak mohou využívat různé aplikace, např. pro správné dělení slov nebo indexování textu.

```
<kapitola xml:lang="cs">
  <para>Celá kapitola je česky.</para>
  <para xml:lang="en">This is the only exception
                      because it is in English.</para>
  <para>Obsah předchozího elmentu byl anglicky.</para>
</kapitola>
```

Jako hodnota atributu se uvádí kód jazyka podle BCP 47[5]. Jazykový kód českého jazyka je `cs`, pro slovenštinu je `sk` a pro angličtinu `en`. Kompletní přehled kódů najdete například na adrese *http://www.loc.gov/standards/iso639-2/langcodes.html*. Lze používat i třípísmenné kódy, což je pro více exotické jazyky dokonce nutnost.

## 1.15.2 xml:space

Jak jsme si již řekli, bílé znaky jsou v dokumentu XML důležité a parser (pokud ho k tomu nepřinutíme) je nijak automaticky neodstraňuje. Záleží pak na konkrétní aplikaci, jak se s bílými znaky dále vypořádá. Mnoho jazyků založených na XML – třeba XHTML nebo XSL-FO – vícenásobné výskyty bílých znaků nahrazují jednou mezerou. Jsou ale případy, kdy se nám to nehodí. XML proto nabízí možnost, jak aplikaci předat informaci o tom, že uvnitř elementu se nemají bílé znaky nijak upravovat. Informace se předává tím, že se k elementu přidá atribut `xml:space` a nastaví se na hodnotu `preserve`. Druhou možností je nastavit jej na hodnotu `default`, kdy se pak uplatní výchozí nastavení aplikace pro práci s bílými znaky.

[5] *http://www.ietf.org/rfc/bcp/bcp47.txt*

Poznamenejme ještě jednou, že atribut `xml:space` slouží k ovlivnění toho, jak se k bílým znakům zachová aplikace, ne parser XML, který dokument načítá.

### 1.15.3 xml:id

V mnoha případech se hodí, když můžeme element jednoznačně identifikovat. V XML k tomu slouží atribut `xml:id`, který elementu přiřadí jednoznačný identifikátor. V jednom dokumentu se přitom nemohou vyskytovat dva elementy se stejným identifikátorem. Atribut `xml:id` tak má podobnou úlohu jako primární klíč v relačních databázích.

Hodnota atributu je přitom poměrně omezená. Identifikátor musí začínat písmenem nebo podtržítkem, za kterým následují další písmena, číslice, tečky, podtržítka nebo pomlčky.

Následující fragment kódu ukazuje použití `xml:id`:

```
<kniha>
  <název>Ze života hmyzu</název>
  <kapitola xml:id="uvod">
    <název>Úvod</název>
    ...
  </kapitola>
</kniha>
```

Na elementy s takto přiřazeným identifikátorem se pak můžeme snadno odvolávat v různých rozhraních a dotazovacích jazycích. Např. rozhraní DOM nabízí metodu `getElementById()`, která vrátí element s daným identifikátorem. Podobně se chová i funkce `id()` v jazyce XPath. Na elementy s identifikátorem se můžeme odvolávat i v XInclude.

### 1.15.4 xml:base

Pomocí atributu `xml:base` jde změnit základní URL, s kterým se skládají relativní URL uvedená v dokumentu. Význam atributu a užití atributu si ukážeme na několika příkladech.

Předpokládejme, že dokument uložený na adrese `http://example.org/manualy/instalace.xml` má následující obsah.

```
<?xml version="1.0" encoding="UTF-8"?>
<manuál xmlns:xi="http://www.w3.org/2001/XInclude">
  <název>Instalační příručka</název>

  <xi:include href="kapitoly/uvod.xml"/>
  <xi:include href="kapitoly/prvni_instalace.xml"/>
  <xi:include href="kapitoly/upgrade.xml"/>

</manuál>
```

Elementy XInclude ukazují na jednotlivé kapitoly. Aby je mohl parser načíst, musí však pro jednotlivé kapitoly znát jejich absolutní URL adresu. Tu získá tak, že relativní adresy z atributu `href` složí se základní adresou dokumentu `http://example.org/manualy/instalace.xml`. Po složení adres získáme následující absolutní adresy jednotlivých kapitol:

```
http://example.org/manualy/kapitoly/uvod.xml
http://example.org/manualy/kapitoly/prvni_instalace.xml
http://example.org/manualy/kapitoly/upgrade.xml
```

V našem dokumentu XML jsme u každého relativního odkazu museli opakovat adresář `kapitoly`, ve kterém byly jednotlivé kapitoly umístěné. Obejít to jde právě pomocí atributu `xml:base`. Tento atribut umožňuje změnit základní URL pro odkazy uvedené v elementu s tímto atributem. Nové základní URL vznikne složením dosavadního základního URL s adresou uvedenou v `xml:base`. Následující příklad ukazuje, jak jsme si ušetřili zápis tím, že jsme základní URL změnili na `http://example.org/manualy/kapitoly/`.

```
<?xml version="1.0" encoding="UTF-8"?>
<manuál xmlns:xi="http://www.w3.org/2001/XInclude"
        xml:base="kapitoly/">
  <název>Instalační příručka</název>

  <xi:include href="uvod.xml"/>
  <xi:include href="prvni_instalace.xml"/>
  <xi:include href="upgrade.xml"/>

</manuál>
```

Atribut `xml:base` nalézá uplatnění nejen ve spojení s elementy XInclude, ale obecně s jakýmikoliv elementy, které vytvářejí nějaký druh odkazů – např. XLink, katalogové soubory apod.