X-definice 3.1

Popis jazyka (předběžná verze)

Autor: **V.Trojan**

Verze: **3.1.003.001**
Ze dne: **21. 3. 2018**

Obsah

Tabulky

Změny dokumentu

Version 3.1.003.007 (26.01.2018)

(1) Metoda Skriptu "getCounter" je nahrazena metodou "getOccurrence" (viz).

Version 3.1.003.005 (06.01.2018)

(1) Implementtován nový typ hodnoty "Locale" ve Skriptu (viz 3.2.3.7*Locale (informace o oblasti)*).

(2) Doplněný metody Skriptu "format" and "printf" (viz *Tabulka* 10 - *Základní metody* a *Tabulka* 23 - *Metody nad objekty typu Output (výstupní stream)*).

Verze 3.1.003.003 (17.12.2017)

(1) doplněna chybějící metoda Skriptu "equalsIgnorCase(s)" pro hodnoty typu String.

Verze 3.1.002.004 (25.09.2017)

(2) deklarace extterních metod přesunuta do <xd:deklaration> pomocí příkazu "external method".

Verze 3.1.002.003 (18.08.2017)

(1) do xd:declaration doplnen atribut xd:scope = ["lolcal", "global"]. Defaultní hodnota je "global".

Verze 3.1.002.001

(1) Výchozí verze tohoto dokumentu.

1 Anotace

Tento dokument je součástí interní projekční dokumentace obecných nástrojů používaných v projektech společnosti Syntea sro. Obsahuje specifikaci technologie, která je nazývána "**X-definice**" a je používána u projektů, ve kterých se pracuje se zprávami a dokumenty XML.

Dokument není přílohou žádné konkrétní projekční dokumentace. Je součástí obecné interně používané technologie, sloužící pro popis a zpracování strukturovaných zpráv a XML dokumentů. Vzhledem k neustálému rozvoji technologie X-definic je i tento dokument průběžně doplňován a upravován.

Text tohoto dokumentu předpokládá, že čtenář je alespoň elementárně seznámen se zápisem pomocí značkového jazyka XML. Pro základní orientaci lze doporučit českou publikaci [1], případně webové odkazy [2] až [4].

X-definice je jazyk, který umožňuje popsat strukturu, obsah, a i zpracování obsahu XML objektů nebo jejich konstrukci. X-definice mohou nahradit existující technologie, užívané pro validaci XML dokumentů, a to jak DTD i XML schémat, případně Schematron nebo v v určitých případech XSLT. Oproti DTD a XML schématům je možné pomocí X-definic propojit validaci XML dokumentu s výkonnými procedurami. Výhodou X-definic je jejic snadná čitelnost, přehlednost a intuitivní srozumitelnost. Programové prostředky pro X-definice byly navrženy tak, aby jimi bylo možné zpracovávat datové soubory s neomezenou velikostí (soubory v reálných projektech mohou mít velikost mnoha GByte).

X-definice je XML dokument, jehož struktura je podobná struktuře popisovaných XML dat. To umožňuje rychle a víceméně intuitivně navrhnout X-definici pro popis XML dat. Významnou výhodou je skutečnost, že X-definice zahrnuje popis pro validaci, zpracování i konstrukci XML dat. Oproti použití XML schémat, Schematronu a XSLT, což vyžaduje různé formy zápisu, je údržba integrovaného zápisu pomocí X-definic mnohem snadnější a přehlednější.

Vlastnosti popisovaných objektů se zapisují do textových hodnot atributů a textových uzlů X-definic pomocí jazyka, který nazýváme „skript“ X-definic. Při návrhu X-definice „krok poí kroku“ lze skript postupně dpolnit o procesní příkazy proí zpracování XML dat nebo jejich konstrukci.

Dotazy, poznámky a hlášení chyb poslílejte na adresu: *xdef@syntea.cz*.

Aktuální verzi X-definic můžete stáhnout na adrese: *http://www.xdef.cz*.

2 Úvodem

Představme si, že máme nějaký XML element, např. popisující záznam o zaměstnanci, který bude mít např. tvar:

```
<Zaměstnanec Jméno              = "Karel"
             Příjmení           = "Novák"
             Nástup             = "12.3.1996"
             Plat               = "21700"
             Kvalifikace        = "elektrikář"
             VedlejšíKvalifikace = "oprava automatiky" />
```

Vidíme, že tento záznam obsahuje údaje různého typu (jména, datum, plat, popis kvalifikace). Chceme-li popsat obsah takového elementu, můžeme tyto údaje nahradit zápisem, zda jsou povinné a případně popsat jejich typ. Formální popis našeho elementu (tzv. **model elementu**) pak může vypadat následujícím způsobem:

```
<Zaměstnanec Jméno              = "required string(1,30)"
             Příjmení           = "required string(1,30)"
             Nástup             = "required datetime('d.M.yyyy')"
             Plat               = "required int(1000,100000)"
             Kvalifikace        = "required string()"
             VedlejšíKvalifikace = "optional string()" />
```

Vidíme, že model elementu se podobá popisovanému elementu, ve kterém jsou původní údaje nahrazeny zápisem ve zvláštním jazyce, který nazýváme **skript X-definic**. Důležité je, že je to XML dokument, který se podobá popisovaným datovým objektům. V **X-definici** je možné pomocí modelů elementů popsat strukturu XML dat, vlastnosti jednotlivých údajů, jejich výskyt i případně popsat jejich zpracování. X-definici je možné použít k popisu XML dat a ke kontrole jejich správnosti (**validaci**) případně, jak uvidíme dále, i popsat jejich **zpracování** nebo **konstrukci** XML nového dokumentu.

2.1 Model elementu

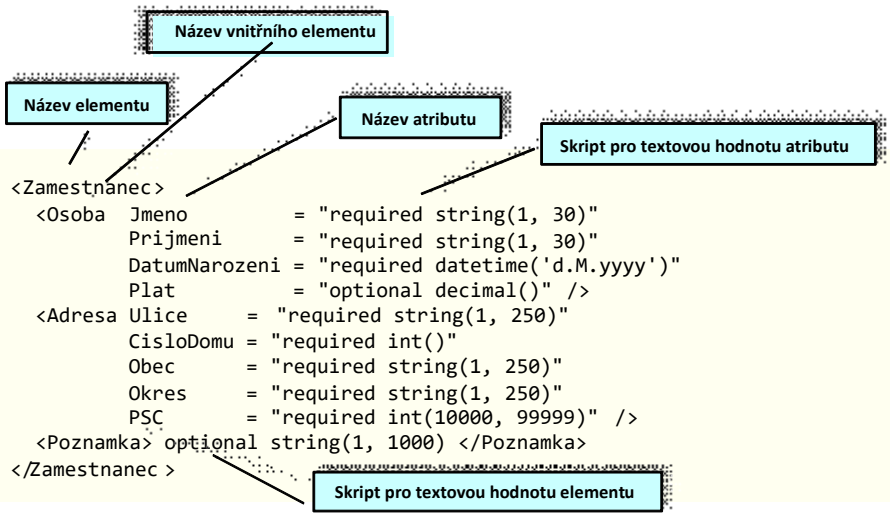
V této kapitole neformálně vyložíme pojmy, se kterými budeme v X-definicích pracovat. V dalších kapitolách pak budou tyto pojmy vysvětleny podrobně.

Základnem X-definic je **model elementu**, pomocí něhož popsujeme strukturu XML elementu a jeho další vlastnosti. Jak jsme ukázali v předešlém odstavci, popis struktury se podobá struktuře původního XML dokumentu.

Reálná data bývají většinou složitější a mají více elementů. Předpokládejme například element, který popisuje zaměstnance takto:

```
<Zamestnanec>
  <Osoba  Jmeno         = "Karel"
          Prijmeni      = "Novák"
          DatumNarozeni = "13.7.1971"
          Plat          = "18.800" />
  <Adresa Ulice     = "Malá"
          CisloDomu = "5"
          Obec      = "Velká"
          Okres     = "Praha-západ"
          PSC       = "23400" />
  <Poznamka>
    Je to velmi dobrý pracovník.
  </Poznamka>
</Zamestnanec>
```

Uvedený příklad obsahuje nejen prosté elementy s atributy, ale také textovou hodnotu uvnitř elementu (`"Je to velmi dobrý pracovník"`). Aby bylo možné popsat textové hodnoty uvnitř XML elementů obdobně jako atributy, zapíšeme skript přímo jako textovou hodnotu tohoto elementu. Popis struktury pro posledně uvedený příklad může mít tvar:



Abychom mohli prohlásit, že dokument je správný (validní), je třeba zkontrolovat nejen výskyt, ale i formální správnost údajů textových hodnot atributů a elementů. Do skriptu atributů a textových uzlů doplníme tedy další údaje o vlastnostech, které musí splňovat. Specifikace **kontroly typu** dat, tvoří spolu s údajem o výskytu tzv. "**validační sekci**" skriptu. Kontrolu typu dat zapíšeme pomocí metody, která kontrolu provede a vrátí informaci podle toho, zda údaj požadovanou vlastnost splňuje nebo nesplňuje. Budeme-li například požadovat, aby nějaký údaj představoval celočíselnou hodnotu, zapíšeme tuto skutečnost ve skriptu metodou "int()" (v tomto případě se jedná o kontrolu, zda hodnota atributu je platný zápis celého čísla). Metody obecně mohou mít uveden seznam parametrů, které dále určují jejich činnost. Např. ve funkci "int()" lze pomocí parametrů zadat minimální a maximální povolenou hodnotu. Zápisem "int(10000, 99999)" se zkontroluje, zda hodnota je číslo v rozsahu 10000 až 99999. Chceme-li, aby nějaký údaj obsahoval řetězec minimálně o délce 2 a maximálně 30 znaků, můžeme to zapsat jako "string(2,30)". Validační metoda "datetime('dd.MM.yyyy')" otestuje datum podle masky v parametru. Implementované kontrolní metody a popis jejich parametrů jsou uvedeny v *Tabulka 7* a *Tabulka 8*.

Jak vidíme, je model elementu rovněž element. V rámci X-definice musí mít model jednoznačné jméno. Je vzorem, na nějž je možno se odkazovat v rámci jedné X-definice nebo i z jiných X-definic nebo může být použit jako odkaz na popis kořenového elementu popisovaných dat.

Model elementu neobsahuje popis výskytu (ten vyplývá z místa, kde je na něj odkaz), nicméně potomci modelu již popis počtu výskytů mít mohou. Ukažme příklad, který popisuje konkrétní bydlící rodinu:

```
<Rodina>
  <Otec  Jmeno         = "Karel"
         Prijmeni      = "Novák"
         DatumNarozeni = "13.7.1971"
         Plat          = "18.800" />
  <Matka Jmeno         = "Jana"
         Prijmeni      = "Nováková"
         DatumNarozeni = "765322/0029"
         Plat          = "11.400" />
  <Syn   Jmeno         = "Karel"
         Prijmeni      = "Novák"
         DatumNarozeni = "9.11.1992" />
  <Syn   Jmeno         = "Jan"
         Prijmeni      = "Novák"
         DatumNarozeni = "1.2.1994" />
  <Dcera Jmeno         = "Pavla"
         Prijmeni      = "Nováková"
         DatumNarozeni = "27.5.1996" />
  <Pobyt Ulice         = "Malá"
         CisloDomu     = "5"
         Obec          = "Velká"
         Okres         = "Praha-západ"
         PSC           = "23400" />
</Rodina>
```

Poznámka: Pro jména speciálních elementů a atributů v X-definicích je používán zápis ve tvaru "xd:jméno" kde xd je prefix jmenného prostoru X-definic a jméno je název speciálního atributu. Tento zápis slouží k odlišení objektů X-definic a datových objektů. V našem textu používáme pro jmenný prostor X-definic důsledně prefix "xd". Uživatel může pro X-definice definovat vlastní jiný prefix, např. "p" pomocí atributu xmlns:p="www.syntea.cz/xdef/3.0" *v záhlaví X-definice. O jmenných prostorech viz [1] a [2]). Kompletní příklad X-definice (tj. celý příslušný XML soubor) je uveden v závěru této kapitoly.*

Chceme-li odkázat na již existující strukturu, můžeme použít **odkaz**, který se zapisuje do skriptu pomocí "**ref**". Modely představují základní kameny, z nichž se skládají X-definice. V jedné X-definici může být popsáno více modelů elementů. Ve skriptu modelu elementu se může pomocí zápisu "ref" vyskytnout **odkaz** na jiný model, informace o výskytu však musí být uvedena a doplní se do kopie modelu z odkazu. Všimněme si, že mezi odkazem a validační částí je znak ";" (středník), který ve skriptu slouží jako oddělovač jednotlivých částí (sekcí skriptu). Středník na konci skriptu je nepovinný, ale může být zapsán. Skript, vztahující se k elementu se zapisuje do zvláštního atributu ze jmenného prostoru X-definic "xd:script". Údaj o minimálním a maximálním počtu výskytů je možné zapsat několika způsoby (např. „optional“ nebo „?“, „1..*“ nebo „+“ - viz 3.2.11 a 3.2.18). V dalším textu budeme zapisovat výskyt spíše zkrácenou formou.

Odkaz na model můžeme zapsat do sekce skriptu označené „ref“ (jednotlivé sekce skriptu jsou oddělené středníkem):

```
    <Syn xd:script = "occurs 0..12; ref Osoba" />
```

Element „Syn“ má strukturu popsanou v elementu „Osoba“ a povolený počet výskytů je minimálně 0 a maximálně 12.

Tímto způsobem je možné vytvářet agregovanější definice. Uvažujme definici rodiny, která může, ale nemusí mít otce, musí vždy mít matku a ne více než 12 dcer a 12 synů. Popis elementu "Rodina" pak bude vypadat takto:

```
<Rodina>
  <Otec  xd:script = "occurs 0..1; ref Osoba" />
  <Matka xd:script = "occurs 1; ref Osoba" />
  <Syn   xd:script = "occurs 0..12; ref Osoba" />
  <Dcera xd:script = "occurs 0..12; ref Osoba" />
  <Pobyt xd:script = "occurs 0..1; ref Adresa" />
</Rodina>
```

Ve zkrácené formě zápisu výskytu můžeme vynechat slovo „occurs“. Oba zápisy „optional“ , „0..1“ můžeme nahradit znakem „?“. Zápis „occurs 1“ nebo „required“ můžeme vynechat – doplní se automaticky. Model „Rodina“ pak můžeme zapsat:

```
<Rodina>
  <Otec  xd:script = "?; ref Osoba" />
  <Matka xd:script = "ref Osoba" />
  <Syn   xd:script = "0..12; ref Osoba" />
  <Dcera xd:script = " 0..12; ref Osoba" />
  <Pobyt xd:script = "?; ref Adresa" />
</Rodina>
```

Z tohoto zápisu je např. patrné, že na daném místě má být element "Syn", je popsán modelem elementu se jménem "Osoba" a může se vyskytnout maximálně 12krát nebo vůbec. V následujícím zápisu počet výskytů elementu "Syn" neomezíme (nemusí se vyskytnout vůbec nebo libovolně krát). Počet opakování zapíšeme tedy jako „*“:

```
<Rodina>
  <Otec  xd:script = "?; ref Osoba" />
  <Matka xd:script = "ref Osoba" />
  <Syn   xd:script = "*; ref Osoba" />
  <Dcera xd:script = "*; ref Osoba" />
  <Pobyt xd:script = "?; ref Adresa" />
</Rodina>
```

Do skriptu můžeme dále doplnit informace o tom, co se má stát, když některý typový výraz nebude splněn a naopak, je-li typ v pořádku. Skutečnost, že nastala v nějakém okamžiku zpracování určitá podmínka, nazýváme **událost**. Každé události definované v X-definici je ve skriptu přiděleno jméno, za něž se píší příslušné akce. Zápis, který začíná jménem události, za nímž následuje příkaz v této události vyvolaný, nazýváme **akce**. Není-li akce popsána, provedou se **standardní akce**, které jsou definovány pro danou událost zvlášť (např. zkopírování údaje do výsledného objektu, hlášení chyby atd.). Událost, kdy typová kontrola proběhne s kladným výsledkem, má jméno **onTrue**, opačná událost má jméno **onFalse**. Mluvíme pak o akci "onFalse", o akci "onTrue" atd. V našem příkladě můžeme například popsat akce, které se provedou na základě kontroly typu atributu "plat":

```
Plat = "? int(1000,50000); onTrue uloz(); onFalse error('chybný plat')"
```

Bude-li údaj správný, vyvolá se uživatelská procedura "uloz", nebude-li, pak se vyvolá procedura "error" s parametrem "chybný plat". Není-li k určité události popsána žádná akce, pak procesor provede akci standardní (např. zápis hodnoty do výsledného XML dokument v případě splněného typového výrazu. V opačném případě se provede hlášení příslušné chyby.

Všimněme si, že zápis znakového řetězce v metodě "error" je ohraničen apostrofy, zatímco celá textová hodnota atributu je uzavřena do uvozovek. To umožňuje přehledný zápis znakových literálů uvnitř hodnot atributů. Použití obou znaků je možné obrátit:

```
Plat = 'int(1000,50000); onTrue uloz(); onFalse error("Chybný plat");'
```

Model elementu představuje popis elementu, obsahující pro jednotlivé objekty skript s validační sekcí a případně s dalšími informacemi (jejich podrobný popis je uveden v 3.2.10). Připomeňme, že v modelu elementu nemá smysl zápis výskytu, protože je uveden jako kořenový prvek.

Čtenář si asi položí otázku, proč jsme zavedli pojmy "struktura elementu" a "model". Důvodem je skutečnost, že mnohdy provádíme transformace XML objektů, ve kterých se mění či transformují hodnoty objektů do výsledného tvaru, který se může lišit od tvaru původního, ale jejich struktura je stejná, nebo podobná, takže je vhodné oddělit popis struktury objektu od jeho obsahu. Např. datum může být na vstupu zapsáno několika povolenými způsoby. Jiným typickým příkladem je zpracování XML objektů v databázích, kdy se mohou např. nahradit některé vstupní hodnoty hodnotami z číselníků apod. Model elementu obsahuje oproti popisu struktury i popis typů a případně akcí v konkrétních instancích objektů. O modelech elementů můžeme pak prohlásit, zda jsou či nejsou odvozeny z určité konkrétní struktury.

Často se vyskytují situace, kdy víme, že skupiny potomků určitého elementu tvoří určité skupiny. Popis skupin se vkládá do zvláštních pomocných elementů (s prefixem „xd“). Jako příklad si uveďme situaci, kdy známe vlastnosti potomků elementu, ale nezáleží na jejich pořadí. K popisu takové situace může sloužit tzv. **neuspořádaná skupina** - "xd:mixed", ve které popíšeme její prvky, které se mohou ve skutečnosti vyskytnout v libovolném pořadí. V následujícím příkladě může být výskyt elementů „Syn“ a „Dcera“ promíchané:

```
<Rodina>
     <Otec  xd:script = "?; ref Osoba"/>
     <Matka xd:script = "ref Osoba" />
     <xd:mixed>
       <Syn   xd:script = "*; ref Osoba"/>
       <Dcera xd:script = "*; ref Osoba"/>
     </xd:mixed>
     <Pobyt xd:script = "ref Adresa" />
</Rodina>
```

Při popisu dat se také může vyskytnout potřeba popsat situaci, kdy se jako prvek smí vyskytnout právě jeden prvek z uvedené množiny. Pro tyto účely slouží tzv. **výběrová skupina** - "xd:choice". Jestliže se smí např. na nějakém místě popisu vyskytnout prvek, popisující buď fyzickou osobu, nebo firmu, můžeme to popsat pomocí výběrové sekce takto:

```
<Subjekt>
   <xd:choice>
       <Osoba xd:script = "ref Osoba"/>
       <Firma nazev     = "string()"
              IC        = "num(8)" />
   </xd:choice>
   <Adresa xd:script = "ref Adresa"/>
</Subjekt>
```

V některých aplikacích je třeba rozlišit několik modelů elementu podle hodnoty atributu. Pro tyto případy lze ve skriptu X-definice popsat akci "match":

```
<Subject>
   <xd:choice>
       <Object xd:script  = "match @Typ EQ 'Osoba'"
               Typ        = "fixed 'Osoba'"
               Jmeno      = "string(1,30)"
               Prijmeni   = "string(1,30)" />
       <Object xd:script  = "match @Typ EQ 'Firma'"
               Typ        = "fixed 'Firma'"
               Nazev      = "string()"
               ICO        = "num(8)" />
   </xd:choice>
   <Adresa xd:script = "ref Adresa"/>
</Subject>
```

Zápis "match @Typ EQ ‘Osoba‘" bude vyhodnocen s hodnotou "pravda", pokud zkoumaný element "Object" obsahuje atribut "Typ" s hodnotou "Osoba". Pokud ano, pak se uplatní první model elementu "Object". V opačném případě se provede vyhodnocení akce "match" na dalším elementu "Object" v pořadí, tj. pro hodnotu ‚Firma‘.

Všimněme si, že akce "match" je logický výraz, který vyhodnotí atributy elementu (není-li specifikace atributu následována relačním operátorem, pak je výsledek pravda, pokud atribut takového jména existuje). Tak lze předchozí příklad přepsat do následující podoby:

```
<Subject>
   <xd:choice>
       <Object xd:script  = "match @Jmeno'"
               Jmeno      = "string(1,30)"
               Prijmeni   = "string(1,30)" />
       <Object xd:script  = "match @ICO"
               Nazev      = "string()"
               ICO        = "num(8)" />
   </xd:choice>
   <Adresa xd:script = "ref Adresa"/>
</Subject>
```

Pozn.: Zápis "@Jmeno" v sekci match vrací hodnotu atributu se jménem "Jmeno" z aktuálně zpracovávaného elementu. Tato hodnota bude "true" pokud atribut existuje a "false" pokud neexistuje. Obdoně je to pro atribut „ICO“.

Poslední typ skupiny je "xd:sequence" - **uspořádaná skupina**. Objekty uvnitř uspořádané skupiny se musí vyskytovat v pořadí, jak jsou popsány. Důvod zavedení uspořádané skupiny je umožnit popsat, že na daném místě v ostatních skupinách není jeden prvek, ale sekvence prvků v určitém pořadí.

```
<Rodina>
  <xd:sequence>
     <Otec  xd:script = "?; ref Osoba"/>
     <Matka xd:script = "ref Osoba" />
     <xd:mixed>
       <Syn   xd:script = "*; ref Osoba"/>
       <Dcera xd:script = "*; ref Osoba"/>
     </xd:mixed>
     <Adresa xd:script = "ref Adresa"/>
   </xd:sequence>
</Rodina>
```

Uspořádaná skupina představuje v tomto případě vlastně popis potomků elementu „Rodina“ (mohli bychom ji vynechat).

Zmiňme se ještě o možnosti zápisu **maker** v X-definici. Pomocí makra můžeme definovat řetězec znaků, kterým bude ve skriptu nahrazen každý odkaz na toto makro (**volání makra**). K deklaraci makra slouží zvláštní typ elementů, které musí být uvedeny na úrovni přímých potomků X-definice:

```
<xd:macro name = "jmeno_makra">string(2,30)</xd:macro>
```

Ve skriptu je možné se na makro odvolat odkazem na jméno makra zápisem" "${jmeno_makra}":

```
<Osoba Jmeno         = "${jmeno_makra}"
       Prijmeni      = "${jmeno_makra}" />
```

Všechna makra se vyhodnotí na začátku před dalším zpracování skriptu. Uvedený příklad má po vyhodnocení maker následující tvar:

```
<Osoba Jmeno         = "string(2,30)"
       Prijmeni      = "string(2,30)" />
```

Nyní můžeme konečně představit celou X-definici. **X-definice** je XML dokument, který má kořenový element s pevně definovaným jménem "xd:def". Potomky kořenového elementu jsou modely elementů a případně deklarace skupin, maker a jiných pomocných elementů, o kterých bude zmínka v dalším textu.

X-definice má přiděleno jméno. Jméno spolu s některými dalšími informacemi zapisujeme pomocí atributů kořenového elementu X-definice, kterému říkáme **záhlaví X-definice**. Jméno X-definice se např. zapisuje do atributu "xd:name". Vzhledem k možnosti výskytu více modelů elementů v rámci jedné X-definice, je třeba do záhlaví X-definice uvést, který model elementu představuje kořenový element, tj. od kterého bude zpracování začínat. Jméno kořenového elementu - volbu odpovídajícího modelu elementu - zapíšeme jako hodnotu atributu "xd:root". Připomeňme, že pro kořenový element je zároveň automaticky použit výskyt „required“. Jestliže jsou v X-definici použity externí uživatelské metody, musí být uveden atribut "xd:methods", ve kterém je seznam Java method v projektu použitých (v našem příkladě je to např metoda "uloz").

Nakonec uveďme náš příklad rodiny jako kompletní X-definici:

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

<!-- X-definice pro element "Rodina" -->
<xd:def  xmlns:xd     ="http://www.syntea.cz/xdef/3.1"
         xd:name      ="Rodina"
         xd:root      ="Rodina" >

<xd:declaration>
  external method {
    void project.Xutils.save() as uloz;
    void project.Xutils.pOtec();
    void project.Xutils.pMatka();
    void project.Xutils.pSyn();
    void project.Xutils.pDcera();
  }
</xd:declaration>

 <!-- Model elementu Osoba -->
 <Osoba  Jmeno        ="string(2,30)"
         Prijmeni     ="string(2,30)"
         DatumNarozeni="dateTime('d.M.yyyy')"
         Plat         ="? int(1000,50000); onTrue uloz(); onFalse error('chyba platu')" />

 <!-- Model elementu Adresa -->
 <Adresa Ulice        ="? string(2,30)"
         CisloDomu    ="string(1,6)"
         Obec         ="string(2,30)"
         Okres        ="string(2,30)"
         PSC          ="int(10000,99999)" />

 <!—Model elementu Rodina -->
 <Rodina>
   <Otec           xd:script="?; finally project.Xutils.pOtec(); ref Osoba" />
   <Matka          xd:script="finally project.Xutils.pMatka(); ref Osoba" />
   <xd:mixed>
      <Syn         xd:script="*; finally project.Xutils.pSyn(); ref Osoba" />
      <Dcera       xd:script="*; finally pDcera(); ref Osoba" />
   </xd:mixed>
   <Pobyt          xd:script="?; ref Adresa" />
 </Rodina>
</xd:def>
```

Metody pOtec, pMatka, pSyn a pDcera jsou implementovány jako statické metody ve třídě project.XUtils,

Je-li naše úloha popsaná pomocí jediné X-definice, není třeba dále popisovat množinu X-definic. Pokud bychom potřebovali k popisu dat více X-definic, je třeba popsat množinu X-definic, na které je možné se vzájemně odvolávat. Více X-definic lze zapsat do elementu "xd:collection" do něhož je možné vložit X-definice jako potomky, případně je možné uvést odkaz na další X-definice či kolekce X-definic např. pomocí atributu "include". Všimněme si, že v uvedeném příkladě jsme oddělili popis elementu "Osoba" a "Adresa" do samostatné X-definice "Popis" a

příslušným způsobem jsme museli upravit odkazy na ně. Odkazem je v tomto případě jméno odkazované X-definice, za nímž následuje znak "#" a jméno referovaného modelu. Náš příklad tedy bude vypadat takto:

```
<?xml version="1.0" ?>
<xd:collection xmlns:xd = "http://www.syntea.cz/xdef/3.1">

<!-- X-definice pro element "Rodina" -->
<xd:def  xd:name   ="Rodina"
         xd:root   ="Rodina" >
 <!-- Model elementu Rodina -->
 <Rodina>
    <Otec       xd:script="?; ref Popis#Osoba" />
    <Matka      xd:script="ref Popis#Osoba" />
    <xd:mixed>
       <Syn     xd:script="*; ref Popis#Osoba" />
       <Dcera   xd:script=" *; ref Popis#Osoba" />
    </xd:mixed>
    <Pobyt      xd:script="?; ref Popis#Adresa" />
 </Rodina>
</xd:def>

<!-- X-definice popisujici modely elementů Osoba  a Adresa -->
<xd:def  xd:name   ="Popis" >

 <!-- Model elementu Osoba -->
 <Osoba Jmeno        ="string(2,30)"
        Prijmeni     ="string(2,30)"
        DatumNarozeni="dateTime('d.M.yyyy')"
        Plat         ="? int(1000,50000)" />

 <!-- Model elementu Adresa -->
 <Adresa Ulice       ="? string(2,30)"
        CisloDomu    ="string(1,6)"
        Obec         ="string(2,30)"
        Okres        ="string(2,30)"
        PSC          ="int(10000,99999)" />
</xd:def>
</xd:collection>
```

3 X-definice

X-definice je XML element "<xd:def>". Každá X-definice má přiděleno jméno, které je zapsáno jako obsah speciálního atributu "xd:name". Jedna X-definice v možině X-definic mít jméno nemusí (nebo je prázdný string). V dalších atributech mohou být zapsány další doplňující informace. Souboru atributů X-definice říkáme **záhlaví X-definice**. Vlastní obsah X-definice je tvořen modely elementů, definicemi skupin a deklarací globálních proměnných a uživatelských metod.

3.1 Model

Modely elementů jsou zapsány jako přímí potomci X-definice a obsahují popis elementů a jejich atributů a potomků. Do X-definice můžeme vnořit více modelů elementů. Každý model elementu v dané X-definici musí mít odlišné jméno (tj. uvnitř jedné X-definice nejsou přípustné dva modely elementu se stejným jménem). Seznam modelů elementů, které mohou vystupovat jako kořen zpracovávaného XML dokumentu je zapsán v záhlaví X-definice v atributu "xd:root". Je-li kořenových elementů více, uvádíme jejich jména oddělená znakem "|" ("svislítko"); např.

```
xd:root="Element1 | Element2"
```

V X-definici se mohou vyskytovat i pomocné modely elementů, které slouží jako vzor pro popis vnitřních elementů a je možné se na ně odkazovat (jejich jména pak nejsou uvedena v atributu "xd:root"). Vlastnosti modelů elementů jsou zapsány pomocí **skriptu** X-definic v pomocném atributu "xd:script", který může být uveden u každého modelu elementu i u vnitřních elementů modelu elementu. Skript, popisující vlastnosti atributů nebo textových uzlů je v modelu elementu zapsán přímo místo hodnot těchto atributů anebo textových uzlů.

3.2 Skript

Skript X-definic je jazyk, který se zapisuje jako textová hodnota do atributů nebo textových uzlů. Pro zápis skriptu elementu je zaveden speciální pomocný atribut "xd:script". Pomocí skriptu se popisují vlastnosti dat a akce, spojené s jejich zpracováním při různých událostech.

Skript má volný formát a je složen z několika částí. Pořadí jednotlivých částí skriptu není závazné a žádná část skriptu není povinná (podle typu události může být doplněna přednastavenou hodnotou). Jako oddělovač jednotlivých částí slouží znak ";" (středník). Zápis skriptu má volný formát, tj. jednotlivé části mohou být odděleny libovolným počtem mezer, tabelátorů nebo řádek. Obecně uvnitř skriptu mohou být uvedeny následující části:

3.2.1.1 Validační sekce.

a) Validační sekce elementu obsahuje specifikaci výskytu elementu. Pokud chybí, doplní se počet výskytu 1 (required).

b) Pro popis textových uzlů elementu nebo atributů může za specifikací výskytu následovat zápis výrazu pro kontrolu typu (výsledkem je hodnota "pravda" nebo "nepravda"). Pokud typová metoda není uvedena, doplní se "string()". Pokud je zpracováván atribut, jehož hodnota je prázdný řetězec a je nastavena volba "ignoreEmptyAttributes" (viz kapitola 3.2.24), pak je atribut ignorován (a pokud je výskyt atributu povinný, vyvolá se pro uvedený atribut s prázdným řetězcem akce "onAbsence").

*Pozn.: X-definice na rozdíl od XML schémat, které popisují vlastnosti textových hodnot elementů je tento popis zapsán přímona místo textové hodnoty. To umožňuje detailněji popsat místo, kde se může textová hodnota vyskytnout. Nicméně uživatel má možnost popsat pomocí pomocného atributu "xd:*text*" textovou hodnotu vzniklou spojením terxtových uzlů elementu. Tímto zápisem jsou pak zpracovány **všechny** textové uzly. Pokud je tato hodnota skriptem změněna a uložena pomocí metody "setText()", pak jsou jednotlivé textové uzly zrušeny a nahrazeny jedinou hodnotou umístěnou jako textová hodnota na konci elementu. Pokud je specifikován atribut "xd:textcontent", pak se před akcí finally pro element vytvoří text spojený ze všech textových uzlů, tento text se přidá na konec seznamu potomků elementu a všechny ostatní textové uzly se odstraní. Příklad:*

```
<Osoba Jmeno    = "string(2,30)"
       Prijmeni = "string(2,30)"
       xd:text  = "string(1,256)" >
  <A …/>
</Osoba>
```

Požadovaný text může být před elementem "A", za ním, nebo dokonce rozdělen na dvě části. Kontrola typu hodnoty se vztahuje postupně na všechny textové uzly zvlášť. Pokud se má kontrola typu vztahovat na všechny textové uzly pospojované dohromady můžeme použít atribut "xd:textcontent". Příkazy sekcí "onTrue", "onFalse" a "finally" se v tomto případě provedou až na konci zpracování celého elementu.

3.2.1.2 Akce (kód volaný, když nastane nějaká událost)

Zápisem akce se určuje, co se má provést v různých událostech v průběhu zpracování objektu. Každý zápis akce je uveden jménem události, za nímž následuje skript s příkazem, který provede v příslušné události (pozn. příkaz akce může být i složený příkaz). Příkazy skriptu jsou podobné jako příkazy v jazyce "C" nebo "Java". Pokud není akce specifikována, je procesorem X-definic provedena standardní akce, příslušející dané události. Jména akcí jsou uvedena v *Tabulka 5 - Události*.

3.2.1.3 Volby (options)

Jejich popis začíná klíčovým slovem "options", za nímž následuje seznam, ve kterém jsou jednotlivé volby odděleny čárkou (popis viz *Tabulka 36 - Volby (options)*). Sekce s volbami může být ve skriptu uvedena nejvýše jednou.

3.2.1.4 Odkaz (reference)

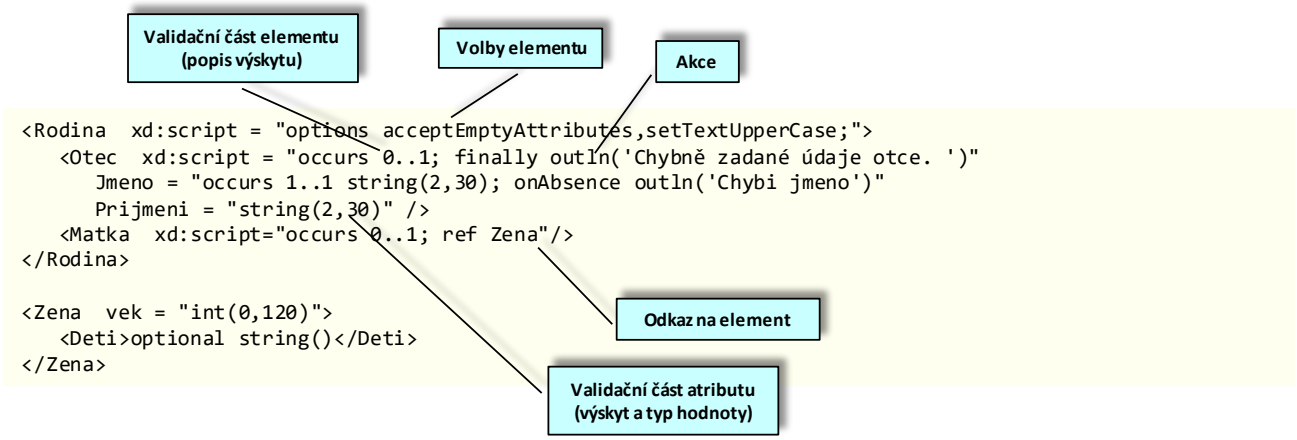
Odkaz na část X-definice, uvedenou na jiném místě. Odkaz převezme hodnoty z místa, na nějž odkazuje. *Pozn. Pokud je ve skriptu odkaz a současně je uvedena i validační sekce, akce, nebo volba, pak jsou odpovídající části v odkazu nahrazeny příslušnými částmi skriptu.*

Do textu skriptu může být vložen také komentář podobně jako v jazycích "C" nebo Java. Komentář se zapisuje mezi znaky "/*" a "*/".

Upozornění: komentáře řádkové uvedené znaky "//", nejsou ve skriptu. Důvod spočívá v tom, že v hodnotách atributů mohou být nové řádky nahrazeny mezerou.

Pořadí jednotlivých shora popsaných částí skriptu je libovolné. Validační sekce je tvořena dvojicí údajů, kde první údaj je specifikace výskytu a druhý údaj, který je nepovinný, je specifikace typu. Je vhodné ji uvést na začátku skriptu.

Ukázka X-definice a skriptu:



Ve výše uvedeném příkladu snadno nalezneme následující sekce:

- **volby** pro element "Rodina", které dovolují uvádět prázdné atributy, resp. převádí znaky textových uzlů na velké symboly
- **validační sekci** elementu "Otec", která dovoluje žádný nebo 1 výskyt daného elementu
- **validační sekci** atributu "Jmeno", která dovoluje přesně 1 výskyt uvedeného atributu s tím, že jeho hodnota bude řetězec o 2 až 30 znacích
- **akce**, které jsou definované pro element "Otec" a atribut "Jmeno". V uvedeném příkladu metody outln vytisknou řádek s textem.

3.2.2 Identifikátory ve skriptu

Ve skriptu se používají identifikátory pro jména X-definic, metod, proměnných, událostí, klíčová slova apod. Identifikátor v skriptu odpovídá tvaru jména elementu nebo atributu v XML. Velká a malá písmena se rozlišují.

Písmena národních abeced jsou přípustná. Jména XML elementů a atributů samozřejmě odpovídají pravidlům pro XML objekty (mohou obsahovat např. znaky ".", "-", ":", tj. tečka, pomlčka a dvojtečka). Nedoporučujeme používat identifikátory s tečkami, dvojtečkami a znakem „-“ (i když to není zakázáno, může dojít k nejednoznačným zápisům).

3.2.3 Typy hodnot proměnných a výrazů ve skriptu

V příkazech skriptu se mohou vyskytnout výrazy a proměnné s hodnotami několika typů. Typy hodnot, které se mohou v skriptu vyskytnout, jsou následující:

3.2.3.1 int - celá čísla

Hodnoty jsou v rozsahu:

```
-9223372036854775808 <= n <= 9223372036854775807
```

Celá čísla se v příkazech skriptu mohou zapsat buď jako dekadická čísla, nebo jako hexadecimální čísla. Hexadecimální zápis čísla musí začínat znaky "0x" nebo "0X", za nimiž následuje posloupnost hexadecimálních číslic nebo písmen ‘a’ až ‘f’ nebo ‘A’ až ’F’.

Zápis zvláštních konstant:

$MININT … minimální hodnota celého čísla (-9223372036854775808).

$MAXINT … maximální hodnota celého čísla (9223372036854775807) .

Aby zápis byl čitelnější, je možné vložit mezi číslice znak „_“ (podtržítko), který nemá vliv na hodnotu čísla. Např. 123_456_789 je ekvivalentní zápisu 123456789.

Pro převod čísla do znakového řetězce je možno použít funkci "toString(hodnota, maska)", kde je možné výstupní formát popsat maskou, kterou představuje řetězec znaků, který obsahuje řídící znaky, které mají následující význam:

- `0` číslice, vedoucí nuly se nahradí mezerou
- `#` číslice, vedoucí nuly se zapíší do výstupu
- `.` způsobí výstup desetinné čárky (tečky)
- `'` prefix a sufix řetězce, který obsahuje řídící znaky, které mají být interpretovány jako znaky výplně (tj. obecný znakový řetězec je uzavřen v apostrofech).

Ostatní znaky tvoří výplň, která je do výstupního řetězce zkopírována.

Příklad:

```
"012" odpovídá masce "##0".
```

3.2.3.2 float - čísla s pohyblivou řádovou čárkou

Hodnoty jsou v rozsahu:

```
-1.7976931348623157E308 <= x <= -4.9E-324
```

nebo

```
4.9E-324 <= x <= 1.7976931348623157E308
```

Zápis čísel s pohyblivou řádovou čárkou odpovídá běžně používanému formátu čísel s pohyblivou řádovou čárkou včetně exponentu. Desetinná čárka se zapisuje vždy jako tečka (bez ohledu na lokální konvence). Exponent můžeme zapsat velkým nebo malým písmenem "e". Při převodu čísla do znakového tvaru pomocí funkce "toString(hodnota, maska)" je možné výstup popsat maskou, kterou představuje řetězec znaků, který obsahuje řídící znaky, které mají následující význam:

`0`	číslice, vedoucí nuly se nahradí mezerou
`#`	číslice, vedoucí nuly se zapíší do výstupu
`E`	odděluje výstup mantisy a exponentu
`.`	způsobí výstup desetinné čárky (tečky)
`' nebo "`	prefix a sufix řetězce, který obsahuje řídící znaky, ale má být interpretován jako výplň (tj. obecný znakový řetězec je uzavřen v apostrofech nebo uvozovkach)

Ostatní znaky masky tvoří výplň, která je do výstupního řetězce zkopírována.

Zápis zvláštních konstant:

$PI	konstanta pi, poměr obvodu kruhu k průměru (3.141592653589793)
$E	konstanta e, základ přirozených logaritmů (2.718281828459045)
$MINFLOAT	nejmenší kladná nenulová hodnota (4.9E-324)
$MAXFLOAT	největší možná kladná hodnota (1.7976931348623157E308)
$NEGATIVEINFINITY	záporné nekonečno
$POSTIVEINFINITY	kladné nekonečno
$NaN	hodnota není validní číslo (Not a Number)

Příklady:

```
"012.00"    odpovídá masce  "##0.00"
"654.32"    odpovídá masce  "##0.00"
"12.30 Kč"  odpovídá masce  "#0.00‘ Kč‘"
".4"        odpovídá masce  "#.0#"
```

3.2.3.3 Decimal - decimální čísla

Decimální čísla s decimálním rozvojem jsou ve Skriptu implementována jako objekty java.math.BigDecimal. Zápis tohoto typu čísla je uveden znaky "0d" nebo "0D", za nimiž následuje zápis celého čísla nebo čísla s pohyblivou řádovou čárkou. V zápisu je možné použít podtržítko, tedy např. "0d123__456_890_999_000_333". Hodnoty typu Decimal je ve výrazech je možné pouze porovnávat, ostatní operace je nutné provádět pomocí příslušných metod.

3.2.3.4 String - znakové řetězce

Řetězce znaků mohou obsahovat libovolné znaky, které jsou přípustné v XML dokumentech. Konstanty se znakovými řetězci se zapisují do apostrofů, nebo do uvozovek (vzhledem k tomu, že hodnoty XML atributů mohou být rovněž uvnitř uvozovek nebo apostrofů, je vhodné uvnitř atributu použít jiný znak, tj. je-li hodnota atributu ohraničena uvozovkami, zapisujeme znakové hodnoty do apostrofů a naopak). Pokud se uvnitř řetězce vyskytuje znak, kterým je řetězec uveden (tj. apostrof nebo uvozovky), uvede se před ním znak ’\’. Výskyt samotného znaku ‘\’ se zapisuje zdvojeně jako ‘\\’. Pomocí znaku "\" lze dále popsat libovolný znak Unicode 16 zápisem "\uxxxx" kde x jsou hexadecimální číslice. Dále lze zapsat některé speciální znaky pomocí následujících dvojznaků:

- `\n` konec řádky - linefeed (LF == `\u000a`)
- `\r` návrat na začátek řádky - carriage return (CR == `\u000d`)
- `\t` tabelátor - horizontal tab (HT == `\u0009`)
- `\f` začátek stránky - form feed (FF == `\u000c`)
- `\b` krok zpět - backspace (BS == `\u0008`)
- `\\` zpetné lomítko (\u005c)

Upozornění: Pokud je text skriptu uveden jako hodnota atributu, XML procesor nahradí všechny výskyty nových řádek mezerami. Proto je třeba do skriptu v atributech zapisovat do znakových řetězců nové řádky jako "\n". Dále je třeba vyvarovat se neúmyslného volání makra. Výskyt dvojice znaků "${" kdekoliv ve skriptu je interpretován jako začátek volání makra a proto je třeba uvnitř znakových řetězců v tomto případě zapsat úvodní znak "$" této dvojice znaků pomocí escape sekvence např. jako "\44" nebo "\u0024", viz 3.2)

3.2.3.5 Datetime - datum a čas.

Hodnoty reprezentující datum a čas. Obsahují rok, měsíc a den. Mohou být zapsány do konstruktoru jako řetězec znaků podle normy ISO8601 nebo mohou být převedeny do vnitřního tvaru pomocí implementované metody "parseDate": např. parseDate('2004-08-10T13:59:05') - viz popis implementovaných funkcí, viz 3.2.21. Doporučený formát zápisu je podle normy ISO8601 (viz [3]), jinak funkce "parseDate" musí mít jako druhý parametr uveden řetězec s maskou, určující formát zápisu. Obdobně lze masku použít jako parametr funkce "toString", např. dat.toString("d.M.yyyy"). Maska je řetězec znaků obsahující řídící znaky pro zpracování vstupních dat a vytvoření objetu s datem (parsování) resp. vytvoření zobrazitelného řetězce z datového objektu (formátování). Ostatní znaky v masce jsou chápány jako znaková konstanta (literál), tj. jejich kopie je vyžadována na vstupu resp. zkopírují se do výstupu. Pokud literál obsahuje písmena, musí se do masky zapsat mezi apostrofy (pokud má být apostrof součástí

literálu, zapisuje se zdvojeně). Opakuje-li se řídící znak, pak při formátování se doplní příslušný počet vedoucích nul a při parsování se přečte zadaný počet cifer. Pro některé řídící znaky má počet opakování jiný význam (viz řídící znaky ‘a’, ‘E’, ‘G’, ‘M’, ‘y’, ‘Z’, ‘z’). Formát navíc může obsahovat následující sekce:

a. **Inicializační sekce**. Inicializační sekce je uzavřena ve složených závorkách "{" a "}".

   Zápis požadované lokalizace (národní jazykově závislé formáty) a přednastavených hodnot. Popis přednastavených hodnot vyžaduje, aby každé hodnotě předcházel příslušný řídící znak. V inicializační části jsou povoleny pouze následující řídící znaky: d, M, y, H, m, s, S, z, Z. Jméno zóny za znakem "z" je uveden v závorkách. Např. {d1M1y2005H16m0s0z(CET)} nastaví datum a čas na 1.1.2005 16:00:00 CET. V závorkách je možné uvést i plný text identifikující místo, např. "Europe/Prague".

   Popis lokalizace je uveden písmenem L za nímž následuje v závorkách identifikátor jazyka a případně může být specifikována i země za čárkou a případně za další čárkou může být specifikována varianta. L(*) nastaví aktuální lokalizaci ze systému. Identifikátory jazyka a země jsou dvou-písmenné a musí odpovídat normám ISO 639 a ISO-3166 (jazyk malými písmeny, země velkými písmeny). Např. L(cs) definuje češtinu. Defaultní hodnota je nastavena na L(en,US) (angličtina, USA), L(es,ES,Traditional_WIN) je španělština, Španělsko, tradiční varianta implementovaná v MS Windows.

b. **Variantní sekce**. Pro parsování je výhodné umožnit více variant formátů. Jednotlivé varianty jsou odděleny znakem ‘|’. Každá varianta má svou vlastní inicializační část. Příklad: Maska d/M/yyyy|yyyy-M-d|{L’en’d MMM yyyy} umožňuje číst data v následujících formátech: 1/3/1999 nebo 1999-0-1 nebo 1 Mar 1999. Varianta má význam pro parsování, pří formátování se použije první varianta.

c. **Nepovinná sekce**. Popis nepovinné sekce je uzavřen v hranatých závorkách "[" a "]". Část uvedená jako nepovinná má význam pouze při parsování a příslušná data mohou ve vstupních datech chybět. Příklad: Maska HH:mm[:ss] odpovídá datům 13:31 nebo 13:31:05. Nepovinné sekce mohou být vnořené (např. HH:mm[:ss [Z]]). Nepovinná sekce má význam pro parsování, pří formátování se ignoruje.

d. **Variantní znaky**. Je-li v masce uveden znak "?" před specifikací řetězce znaků ukončeného apostrofem "‘", pak při parsování může být na daném místě kterýkoliv ze znaků v řetězci (např. d?/.’m?/.’yyyy je platná pro datum ve tvaru 1.3.1999 nebo pro 1/3/1999).

e. **Řídící znaky masky**. Parser a formátovač datumu interpretuje hodnoty datumu podle řídících znaků uvedených v následující tabulce:

Tabulka 1 - Řídicí znaky v datumové masce

Znak	Typ	Popis	Příklad
a (a více a)	text	Informace o části dne	AM, PM
D	číslo	Den v roce bez vedoucí nuly	4
DDD	číslo	Den v roce s vedoucími nulami	004
d	číslo	Den v měsíci bez vedoucí nuly	9
dd	číslo	Den v měsíci s vedoucí nulou	09
E, EE, EEE	text	Zkrácený den v týdnu	Mon, Tue...
EEEE (a více E)	text	Plný název dne v týdnu	Monday
e	číslo	den v týdnu jako číslo(1 = Po, 7 = Ne) bez vedoucí nuly	3
ee (a více e)	číslo	den v týdnu jako číslo(1 = Po, 7 = Ne) s vedoucí nulou	03
G (a více G)	text	Označení éry	AD, BC
H	číslo	Hodina v intervalu 0 – 23 bez vedoucí nuly	8
HH	číslo	Hodina v intervalu 0 – 23 s vedoucí nulou	08
h	číslo	Hodina v intervalu 1 – 12 bez vedoucí nuly	9
hh	číslo	Hodina v intervalu 1 – 12 s vedoucí nulou	09
k	číslo	Hodina v intervalu 0 – 11 bez vedoucí nuly	9
kk	číslo	Hodina v intervalu 0 – 11 s vedoucí nulou	09
K	číslo	Hodina v intervalu 1 – 24 bez vedoucí nuly	9
KK	číslo	Hodina v intervalu 1 – 24 s vedoucí nulou	09
M	číslo	Číslo měsíce v roce bez vedoucí nuly	6
MM	číslo	Číslo měsíce v roce s vedoucí nulou	06

MMM	text	Zkrácený název měsíce	Jan
MMMM	text	Úplný název měsíce	January
m	číslo	Počet minut (bez vedoucí nuly)	1
mm	číslo	Počet minut (s vedoucí nulou)	01
RR	číslo	Rok dvojmístný (podle Oracle). Století se doplní podle následujících pravidel: Je-li RR v rozsahu 00 .. 49, pak a) jsou-li poslední dvě číslice aktuálního roku 00..49. pak se první číslice doplní ze současného století. b) jsou-li poslední dvě číslice aktuálního roku 49..99, pak se první číslice doplní ze současného století zvětšeného o jedničku. Je-li RR v rozsahu 50..99, pak c) jsou-li poslední dvě číslice aktuálního roku 00..49, pak se první číslice doplní ze současného století zmenšeného o jedničku. d) jsou-li poslední dvě číslice aktuálního roku 49..99, pak se první číslice doplní ze současného století.	
S	číslo	Počet sekund (bez vedoucí nuly)	5
ss	číslo	Počet sekund (s vedoucí nulou)	05
yy	číslo	Rok ve dvojciferném tvaru, který je interpretován tak, že hodnotám "01" až "99" jsou přiřazeny hodnoty 1901 až 1999 a hodnotě "00" je přiřazena hodnota 2000	
y	číslo	Rok (bez vedoucích nul	1
yyyy (a více y)	číslo	Rok v čtyřmístném (resp. vícemístném) tvaru	1989
Y	číslo	Rok podle specifikace ISO (může být i záporný a mít vedoucí nuly)	
YY	číslo	Dvojciferný rok, století se doplní z aktuálního data.	
z	text	Zkrácené jméno zóny	CEST
zz(a více z)	text	Úplné jméno zóny	Central European Sumer Time
Z	zóna	Zóna ve tvaru + nebo – následované HH:mm	+01:00
ZZ	zóna	Zóna ve tvaru + nebo – následované H:m	+1:0
ZZZZZ	zóna	Zóna ve tvaru + nebo – následované HHmm	+0100
ZZZZZZ (a více Z)	zóna	Zóna ve tvaru + nebo – následované HH:mm (jako Z)	+01:00

Příklady datumu a odpovídající masky:

```
"Dne 3.4.2003 v 7:05"      odpovídá masce  "'Dne 'd.M.yyyy v H:mm"
"2003-04-30 17:05T+01:00"  odpovídá masce  "yyyy-MM-dd HH:mm[:ss]‘T‘[Z]"
```

3.2.3.6 boolean - logické hodnoty

Možné hodnoty jsou "pravda" a "nepravda". Mohou být zapsány pomocí identifikátorů "true" a "false".

3.2.3.7 Locale (informace o oblasti)

Tento typ obsahuje informace o jazyce, zemi a zeměpisné, politické nebo kulturní oblasti. Může se použít, když jsou informace pro tisk vytvořeny z datových hodnot (formát čísla, měny, data a času atd.). Hodnota Locale může být vytvořena podle následujících konstruktorů:

new Locale(language) nebo

new Locale(language, country) nebo

new Locale(language, country, variant)

kde "language" je dvě malá písmena odpovídající kódu jazyka podle ISO-639, "country" jsou dvě velká písmena odpovídající kódu země podle ISO-3166 a "variant" je string s textem odpovídajícmí specifickému kódu podle dodavatele resp. prohlížeče.

3.2.3.8 Regex - regulární výrazy

Objekty tohoto typu lze vytvořit pomocí konstruktoru new Regex(s), kde s je string se zdrojovým tvarem regulárního výrazu. Regulární výraz odpovídá specifikaci podle XML schémat.

3.2.3.9 RegexResult - výsledek regulárního výrazu

Objekty tohoto typu vzniknou jako výsledek metody "getMatcher".

3.2.3.10 Input/Output - streamy, soubory

Objekty tohoto typu slouží k práci se soubory a streamy. Automaticky jsou předdeklarovány dva globální objekty typu "Output" – "$stdOut" (odpovídá java.lang.System.out) a "$stdErr" (odpovídá java.lang.System.err) a jeden soubor "$stdIn" typu "Input" (odpovídá java.lang.System.in). Soubory mohou být použity jako první parametr metody "putReport". Soubor "$stdOut" je použit automaticky v metodách "out", "outln" (tj. parametr nemusí být specifikován). Obdobně je použit soubor "$stdErr" v metodě "putReport".

3.2.3.11 Element – XML elementy

Objekty tohoto typu odkazují na hodnotu typu "org.w3c.dom.Element". Mohou být např. výsledkem metod "getElement", nebo metod nad objekty typu "Container".

3.2.3.12 Bytes – pole bytů

Tento objekt může být výsledkem metody parseBase64 nebo parseHex. Konstruktor pro vynulované pole bytů je:

```
Bytes bb = new Bytes(10); /* Vynulované pole 10 bytů. */
```

Metody pro práci s poli bytů viz 3.2.21.

3.2.3.13 NamedValue – pojmenovaná hodnota

Pojmenovaná hodnota je dvojice, kterou tvoří jméno a hodnota – ta může být kterýkoliv z uvedených typů hodnot. Jméno musí odpovídat XML jménu. Vytvořit pojmenovanou hodnotu můžeme zápisem začínajícím znakem "%" následovaným bezprostředně jménem, za nímž následuje rovnítko ("=") a výraz s hodnotou. Např.:

```
NamedValue nv = %x:y..named-value = "ABC";
```

3.2.3.14 Container – sekvence a mapa

Objekty tohoto typu mohou být výsledkem některých metod (xPath, XQuery atd). Obsah tohoto typu se skládá z mapované části, která obsahuje pojmenované hodnoty a seznamu prvků (sekvenční část), jehož prvky mohou být kterýkoliv ze shora uvedených typů kromě pojmenované hodnoty. Prázdný Container můžeme vytvořit konstruktorem

```
Container c = new Container();
```

Poznámka: z důvodu kompatibility se staršími verzemi je možné použít jako jméno typu i identifikátor „Context“.

Hodnotu typu Container můžeme také zapsat mezi znaky "[" a "]", kde uvedeme seznam hodnot. Pojmenované hodnoty se uloží do mapované časti a nepojmenované hodnoty vytvoří seznam hodnot. Např.:

```
Container c = [%a=1, %b=x, p, [y,z], "abc"];
```

Mapovaná část bude obsahovat pojmenované hodnoty "a" a "b". Sekvenční část je seznam hodnotou p, vnořeným objektem Container a stringem "abc".

K práci s objektem Container je možné použít řadu metod, uvedených níže (např. toElement).

Container se může vyskytnout také v ve výrazech typu boolean (tj. může být v sekci „match“, v příkazu „if“ atd). Hodnota objektu Container se převede na boolean podle následujících pravidel:

1. Obsahuje-li objekt právě jednu sekvenční položku typu boolean, pak je výsledek shodný s hodnotou této položky.

2. Obsahuje-li právě jednu sekvenční položku typu typu int, float nebo BigDecimal, pak je výsledek true, pokud je hodnota této položky různá od nuly.

3. Ve všech ostatních případech je true, pokud je sekvenční část objektu neprazdná, jinak je výsledek false.

Poznámka: Typ Container *je výsledkem výrazů XPath a XQuery. Pokud je XML uzel, nad kterým se výraz má provést null, vrátí se prázdný* Container.

3.2.3.15 Exception – výjimka

Tento objekt je předán při zachycení programové výjimky (chyby) v konstrukci "try {…} catch(Exception(ex)) { … }". Výjimku lze způsobit i příkazem skriptu "throw". Objekt typu výjimka je možné ve skriptu vytvořit pomocí konstruktoru "new Exception(text_chybové_zprávy)".

3.2.3.16 Parser – nástroj ke kontrole typu

Tento objekt provádí typovou kontrolu textových hodnot v XML dokumentu.

3.2.3.17 ParseResult – výsledek parsování

Objekty tohoto typu vzniknou jako výsledek parseru. Pokud se tento objekt vyskytne ve výrazu boolean, převede se na boolean a hodnota je true, pokud v objetu nejsou indikovány chyby, jinak je hodnota false (t.j. doplní se automatické volání metody "matches()").

3.2.3.18 Report - zpráva

Tento typ objektu představuje parametrizovatelnou a jazykově přizpůsobitelnou zprávu. Zprávu můžeme vytvořit:

```
Report r = new Report("MYREP001", "Toto je chyba");
```

případně ji můžeme získat např. metodou "getLastError".

3.2.3.19 BNFGrammar - BNF gramatika

Tento typ objektu je definován pomocí zvláštní deklarace. BNF gramatika se deklaruje ve zvláštním deklaračním elementu nebo jako proměnná. Viz kapitola 5.1, ve kterém je popis prostředků BNF.

3.2.3.20 BNFRule - pravidlo BNF gramatiky

Odkaz na pravidlo z gramatiky. Pravidlo gramatiky lze použít např. k validaci textové hodnoty atributů nebo textových uzlů. Pravidlo lze z objektu typu BNFGrammar získat pomocí metody rule(jmeno_pravidla).

3.2.3.21 uniqueSet - tabulka unikátních hodnot

Tento typ je použit pro zajištění unikátnosti hodnot v XML dokumentu a používá se pouze ve spojení s kontrolou typu textových hodnot v XML.

3.2.3.22 Service - služba databáze

Objekt umožňující přípojení ke službám různých databází. Většinou se předává do X-definic pomocí externí proměnné. Nicméně je možné ve Skriptu použít i konstruktor:

```
Service connection = new Service(s1,s2,s3,s4);
```

Parametr s1 je druh databáze (např "jdbc"), s2 je URL databáze, s3 je uživatelské jméno a s4 je heslo.

3.2.3.23 Statement - příkaz databáze

Objekt obsahuje připravený databázový příkaz. Je možné jej vytvořit z objektu Service např. příkazem prepareStatement(s), kde "s" je string s databázovým příkazem:

```
Statement stmt = connection.prepareStatement(s);
```

3.2.3.24 ResultSet - výsledek databázového příkazu

Objekt, obsahující výsledek databázového příkazu. V případě relační databáze obsahuje tabulku, jejíž řádky mají pojmenované sloupce. Řádky je možné postupně zpřístupnit metodou next(). V případě XML databáze závisí výsledek na příkazu, např. to může být objekt Container.

3.2.3.25 XmlOutStream – datový kanál pro průběžný zápis XML objektů

Tento typ objektu umožňuje zápis XML dokumentů, jejichž rozsah by mohl přesáhnout velikost paměti počítače. Často se používá tento způsob zápisu ve spojení s příkazem "forget". Objekt je možné vytvořit konstruktorem "new XmlOutputStream(p1, p2, p3)". Parametr p1 je povinný a musí odpovídat cestě a jménu souboru, do kterého bude zápis proveden. Parametr p2 je jméno znakové tabulky a parametr p3 udává, zda se má vytvořit záhlaví XML dokumentu. Příklad typického použití ve skriptu X-definic:

```
  XmlOutStream xstream=new XmlOutStream("c:/data/file.xml", "UTF-8", true);
...
  xstream.writeElementStart();
  xstream.writeElement(); // write whole child
...
  xstream.writeElement();
  xstream.writeElementEnd(); // write end of started element
...
  xstream.close();
```

3.2.3.26 Reference na atribut aktuálního elementu

Ve skriptu elementů a jejich atributů je možné zapsat "@jméno_atributu". Pokud je tento zápis objeví ve výrazu typu boolean, pak je hodnota true, pokud atribut s tímto jménem existuje, jinak je hodnota false. Pokud je tento zápis uveden ve výrazech se stringem, je výsledek hodnota atributu, nebo prázdný řetězec.

```
<A xd:script = "match (@a AND @b OR @c)" ...
```

Výsledek sekce "match" bude true, pokud v elementu "A" existují oba atributy "a" i "b", nebo atribut "c".

3.2.4 Přístup k hodnotám ve zpracovávaném dokumentu

Ve skriptu X-definic je také možné použít hodnoty atributů či textových uzlů získané ze zpracovávaného XML dokumentu např. pomocí funkcí "getText", "getElement", “getElementText“ (viz *Tabulka 10 - Základní metody*).

3.2.5 Lokální proměnné skriptu

Ve skriptu X-definic je možné v příkazovém bloku deklarovat a používat hodnoty, uložené v lokálních proměnných. Lokální proměnná je reprezentována jménem (identifikátorem), který musí začínat písmenem, za nímž mohou následovat písmena nebo číslice a kdekoliv v identifikátoru může být použit znak "_" (podtržítko). Lokální proměnná musí mít specifikován typ hodnoty, kterou představuje. Platnost lokální proměnné je ukončena příkazem (blokem), v němž byla deklarována.

3.2.6 Proměnné modelu elementu

Ve skriptu elementu je možné deklarovat proměnné, které jsou platné (a tedy přístupné) jen v době zpracování aktuálního elementu. Pokud chceme takové proměnné deklarovat, zapíšeme je do sekce skriptu „var“, která musí být uvedena na začátku:

```
<A xd:script="var int b=0,c=0; occurs *; finally outln('B=' + b + ', C=' + c)">
  <B xd:script="*; finally b++"/>
  <C xd:script="*; finally c++"/>
</A>
```

Na konci zpracování elementu A se vypíše počet výskytů elementů B a C.

Pokud je potřeba uvést více deklaračních příkazů, musí se vložit do svorkových závorek:

```
<A xd:script="var { int i; String s; }; *">
```

3.2.7 Deklarace proměnných a metod

Proměnné a metody je možné deklarovat v elementu "<xd:declaration>", který musí být přímým potomkem X-definice. Na takto deklarované proměnné a metody je možné odkazovat na kterémkoliv místě skriptu z libovolné X-definice. Pokud je však uveden atribut "xd:scope='local'", jsou tyto proměnné a metody viditelné pouze z X-definice, ve které byly deklarovány. Před každou metodou musí být uvedeno jméno typu výsledku, za nímž následuje jméno uživatelské metody, dále následuje deklarace seznamu parametrů a příkazová část metody, která je zapsána do složených závorek "{" a "}". Hodnota výsledku metody se předává příkazem "return". Metody pro validaci textových hodnot musí vrátit hodnotu ParseResult nebo logickou hodnotu "true" nebo "false". Metody určené pro událost "create" v elementech mmohou vrátit např. hodnotu typu "Container", metody pro akci "create" v textových objektech musí vrátit hodnotu typu "String".

Deklarací metod je možné "zčitelnit" a zpřehlednit zápis skriptu.

Zápis metod je podobný jako v jazyce C nebo Java. Formální seznam parametrů uživatelských metod je zapsán v závorkách. Jednotlivé parametry jsou odděleny čárkou. Každý parametr je zapsán jako dvojice, složená ze jména typu a jména parametru. Seznam parametrů může být prázdný, pak se zapíše jako "()". Příkazy uživatelských metod jsou zapsány do bloku ohraničeného závorkami "{" a "}". Parametry mohou být použity v příkazové části metody podobně jako proměnné, jejichž počáteční hodnota je určena při volání metody. Jména typů hodnot, které mohou být použity jako parametry uživatelských metod, jsou v následující tabulce (velikost písmen je nutné dodržet).

Tabulka 2 - Jména typů hodnot parametrů

Jméno typu	Popis typu hodnoty
`Boolean`	Logická hodnota
`Datetime`	Datum a čas
`Float`	Číslo v pohyblivé řádové čárce
`Int`	Celé číslo
`String`	Znakový řetězec
`Regex`	Regulární výraz
`RegexResult`	Výsledek regulárního výrazu
`Output`	Výstupní soubor
`Input`	Vstupní soubor
`Bytes`	Pole bytů (výsledek metody parseBase64)
`Container`	Pole objektů ostatních typů
`Exception`	Výjimka (jen jako parametr konstrukce "catch")
`Message`	Zpráva
`BNFGramar`	BNF gramatika
`BNFRule`	Pravidlo BNF gramatiky
`Object`	Uživatelský objekt

Pozn.: Nezaměňujte jména jmen typů s obdobně pojmenovanými funkcemi pro validaci typů, jejichž jména se typicky liší velikostí počátečního písmena.

V následujícím příkladě jsou metody určené k validaci. Tyto metody musí tedy vrátit hodnotu "true" nebo "false" pomocí příkazu "return". Všimněte si, že pokud tělo kontrolní metody vrací výsledek některé z implementovaných kontrolních metod, může být použito i příslušné hlášení chyby (viz metoda "barva"). Příkaz "return error ('...');" vrací hodnotu "false", protože tuto hodnotu vrací metoda "error"). Proměnná today obsahuje datum a čas začátku zpracování, metoda getToday() vrátí toto datum v daném formátu. Skript je vhodné zapsat do CDATA sekce (kvůli zápisu znaků "<", ">", "&"):

```
<xd:declaration xd:scope="global">
<![CDATA[

Datetime today = now(); /*datum a čas začátku zpracování */

/* vrati datum a cas zacatku zpracovani */
```

```
String getToday() {
  return today.toString("dd.MM.yyyy HH:mm:ss");
}

/* Kontrola udaje se jmenem */
boolean jmeno() {
   if (string(5,30)) {
      outln('Jméno: ' + getText());
      return true;
   }
   return error('Chyba v delce jmena:' + length(getText()));
}

/* Kontrola udaje s barvou */
boolean barva() {
   /* hlaseni chyby provede procedura list, která rovněž vrátí hodnotu */
   return enum('cervena', 'zelena', 'modra');
}

/* Hodnota může být -1 nebo číslo v daném intervalu */
boolean hodnota(int min, int max) {
   if (!int) {
      return error('Nenumericka hodnota');
   }
   i = parseInt(getText());
   if (i == -1)
      return true;
   if (i < min)
      return error('Prilis mala hodnota');
   else if (i > max)
      return error('Prilis velka hodnota');
   return true;
}
]]>
</xd:declaration>
```

3.2.8 Předdefinované proměnné a konstanty

Ve skriptu je možné použít řadu předdefinovaných proměnných a konstant. Předdefinované proměnné a konstanty začínají znakem "$":

Tabulka 3 - Předefinované proměnné a konstanty

Jméno	Typ	Význam
$stdIn	Input	Standardní vstupní soubor
$stdErr	Output	Standardní soubor s chybovým protokolem
$stdOut	Output	Standardní výstup soubor
$PI	Float	Konstanta s číslem nejbližším pi (poměr obvodu kruhu k průměru: 3,14159265 ….)
$E	Float	Konstanta s číslem nejbližším e (základ přirozených logaritmů: 2.71828182….)
$MAXINT	Int	Konstanta s největším celým číslem, použitelným ve skriptu
$MININT	Int	Konstanta s nejmenším celým číslem, použitelným ve skriptu
$MAXFLOAT	Float	Konstanta s největším číslem v pohyblivé řádové čárce, použitelným ve skriptu
$MINFLOAT	Float	Konstanta s nejmenším číslem v pohyblivé řádové čárce, použitelným ve skriptu
$NEGATIVEINFINITY	Float	Konstanta odpovídající zápornému nekonečnu v pohyblivé řádové čárce
$POSITIVEINFINITY	Float	Konstanta odpovídající kladnému nekonečnu v pohyblivé řádové čárce

3.2.9 Výrazy

Ve skriptu je možné získat hodnoty jako výsledek výrazů, které jsou podobné výrazům v jiných programovacích jazycích (Java, C apod.). Podrobný popis přesahuje rámec této publikace a čtenář se s ním může seznámit např. v [5].

Výsledkem výrazu je vždy hodnota některého z výše uvedených typů. Příklad použití výrazu v parametru metody (s odkazem na hodnotu globální proměnné "max"):

```
<Zbozi pocet = "required int(0, max + 100)" />
```

Protože se musí v textech v XML zapisovat znaky "&", "<", ">" pomocí předdefinovaných entit "&", "<" ">", jsou pro některé operátory definována alternativní klíčová slova, která umožňují čitelnější zápis skriptu:

Tabulka 4 - Klíčová slova pro alternativní zápis operátorů XML

Operátor	Alternativní klíčové slovo	Význam
&	AND	Logické "a"
&&	AAND	Podmíněné logické "a"
|	OR	Logické "nebo"
||	OOR	Podmíněné logické "nebo"
<	LT	Relace "menší než"
>	GT	Relace "větší než"
<=	LE	Relace "menší nebo rovno"
>=	GE	Relace "větší nebo rovno"
==	EQ	Relace "rovno"
!=	NE	Relace "nerovno"
<<	LSH	Bitový posun čísla vlevo
>>	RSH	Bitový posun čísla vpravo
>>>	RRSH	Bitový posun čísla vpravo bez znaménka
%	MOD	Aritmetické modulo
^	XOR	Exkluzivní logické "nebo"
!	NOT	Logická negace
~	NEG	Bitová negace (čísla)

Příklad:

```
i = p GE 125 AAND q LT 3;
```

je ekvivalentní zápisu:

```
i = p >= 125 && q < 3;
```

3.2.10 Události a akce

Zápis akce vždy začíná jménem události, za níž následuje příkaz provádějící příslušnou akci (může to být i prázdný příkaz ";").

Tabulka 5 - Události

Jméno události	Popis události
`create`	Událost nastává jen v režimu kompozice v okamžiku, když se zahajuje zpracování nového objektu při průchodu X-definicí (ještě před událostí "`init`"). Příkaz této akce je výraz, jehož výsledkem je odpovídající objekt (element, atribut nebo textová hodnota - obvykle se jedná o vyhledání objektu ve vstupních datech nebo o vytvoření nového objektu). U atributů a textových uzlů je jako výsledek příkazu očekáván textový řetězec a u elementů se očekává objekt typu element (jméno a předci tohoto elementu jsou pro další zpracování nevýznamné). Není-li akce specifikována, provede se převzetí odpovídajícího objektu ze vstupního objektu (podle odpovídající pozice právě zpracovávaného objektu v X-definici). Typ výsledku výrazu musí být jeden z následujících: 1. `org.w3c.dom.Element` nebo `org.w3c.dom.NodeList`, je-li akce součástí skriptu elementu. 2. `String`, je-li akce součástí skriptu atributu nebo textového uzlu.
`default`	Událost je definována pouze pro atributy a textové hodnoty. Pokud atribut nebo textová hodnota neexistuje, vyvolá se akce, jejímž výsledkem je string a jeho hodnota se doplní jako hodnota příslušného atributu nebo textové hodnoty. Událost nastává po zpracování událostí onFalse a onAbsence.

match	Událost pro akci "match" nastává před dalším zpracováním elementu nebo atributu. Tato akce musí vrátit hodnotu "true" nebo "false". Pokud je hodnota "true", pak zpracování pokračuje normálně, pokud je "false", pak aktuální element není zpracován podle modelu elementu, v němž je akce „match“ zapsána. Akce vyvolaná v této události má k dispozici dosud zpracovanou část XML dokumentu včetně atributů (jejich kontrola se však provede, pokud je hodnota match true).
finally	Událost pro akci "finally" nastane na konci zpracování elementu podle daného modelu. Po ní už následuje pouze vymazání z paměti (v případě akce "`forget`").
forget	Akce události "forget" se vyvolá se na konci zpracování elementu (i po akci "finally"). Tato akce má význam především při zpracování velkých souborů s XML daty, které nelze umístit do paměti počítače. Akce "forget" způsobí, že se příslušný objekt po zpracování a po všech kontrolách a akcích (i po akci "finally") odstraní z paměti. Příznaky o provedených kontrolách a čítač výskytu elementu však zůstanou nastaveny. UPOZORNĚNÍ: akce forget se (na rozdíl od ostatních akcí) NEDĚDÍ z referovaných objektů, je ji třeba vždy specifikovat u příslušného elementu!
init	Inicializační akce, která se provede před dalším zpracováním objektu. Automatické provedení některých akcí je možné nastavit pomocí specifikace "options" (viz `ignoreAttrWhiteSpaces, trimAttr, ignoreTextWhiteSpaces, trimText, setAttrLowerCase, setAttrUpperCase, setTextLowerCase, setTextUpperCase`). U elementů je v události "init" při režimu parsování dostupný element, který ještě nemá žádné potomky a není zařazen do stromu výsledného dokumentu, ale již existují všechny (nezpracované) atributy.
onAbsence	Událost nastane, když není splněna podmínka minimálního výskytu elementu nebo když požadovaný objekt chybí. Není-li akce specifikována a není splněna podmínka minimálního výskytu, provede se hlášení chyby do souboru s protokolem o zpracování.
onExcess	Akce události "onExcess" se vyvolá, když element překračuje horní hranici povoleného počtu výskytů. Není-li událost specifikována, provede se hlášení chyby do souboru s protokolem o zpracování a objekt se nezařadí do výsledku.
onFalse	Akce události "onFalse" se provede, je-li výsledek vyhodnocení typu "`false`". Není-li událost specifikována, provede se hlášení chyby do souboru s protokolem o zpracování.
onIllegalAttr	Akce se provede v případě výskytu nedefinovaného nebo nepovoleného atributu. Není-li událost specifikována, provede se hlášení chyby do souboru s protokolem o zpracování.
onIllegalElement	Akce se provede v případě výskytu nedefinovaného nebo nepovoleného elementu. Není-li událost specifikována, provede se hlášení chyby do souboru s protokolem o zpracování.
onIllegalText	Akce se provede v případě výskytu nedefinovaného nebo nepovoleného textu. Není-li událost specifikována, provede se hlášení chyby do souboru s protokolem o zpracování.
onIllegalRoot	Popis této události je povolen pouze v skriptu X-definice. Akce se provede, když v X-definici není nalezen příslušný element popsaný atributem "`xd:root`". Pokud není specifikována, provede se hlášení chyby do souboru s protokolem o zpracování a další zpracování nepokračuje.
onStartElement	Akce události se provede po zpracování seznamu atributů, ale ještě před zpracováním vnitřních uzlů elementu.
onTrue	Akce události se provede, je-li výsledek vyhodnocení typu (výsledku validační akce) "true". Není-li akce specifikována, uloží se kopie objektu do výsledku.
onXmlError	Popis této událost je povolen pouze v skriptu X-definice. Akce se provede, když parser zdrojového XML dokumentu odhalí chybu ve formátu XML. Pokud událost není specifikována, provede se hlášení chyby do souboru s protokolem o zpracování nebo, pokud jde o vážnou chybu, další zpracování nepokračuje a program končí výjimkou SException.

3.2.11 Specifikace výskytu textových hodnot a atributů

Validační sekce ve skriptu textových hodnot se skládá ze specifikace výskytu a kontroly typu. Specifikace výskytu je povinná, specifikace kontroly typu hodnoty je povinná, a pokud není uvedena, může být hodnota jakýkoliv řetězec znaků (i prázdný).

1. Specifikace výskytu je jedno z následujících klíčových slov:
   "required" - atribut nebo textový uzel je povinný (možno zapsat také jako 1 nebo 1..1)
   "optional" - atribut nebo textový uzel je nepovinný (možno zapsat také jako „?“ nebo 0..1)
   "ignore"- atribut nebo textový uzel se sice může vyskytnout, ale je v dalším zpracování ignorován

"illegal" - atribut nebo textový uzel je zakázaný.

Poznámka: z důvodů kompatibility je možno před zápis výskytu zapsat klíčové slovo „occurs“. Specifikace „required“ je defaultní a je možné ji vynechat.

2. Specifikace kontroly typu představuje jméno metody, která provádí kontrolu typu. Tato metoda musí mít uveden seznam parametrů v závorkách, kde jednotlivé parametry jsou odděleny čárkou (prázdný seznam parametrů se zapisuje jako "()").
3. "fixed" - atribut nebo textový uzel je povinný a má pevně stanovenou hodnotu. Zápis této hodnoty je typu znakový řetězec, je povinný a musí následovat za klíčovým slovem "fixed". Zápis slouží ke zvýšení přehlednosti. Zápisem se říká, že na daném místě je povinná uvedená hodnota. Pokud není přítomná ve vstupních datech, doplní se. Pokud se v datech hodnota od uvedené liší, ohlásí se chyba a hodnota se nastaví na uvedenou hodnotu. Tedy skript, obsahující zápis "fixed":

```
fixed '2.0'
```

je totožný se zápisem:

```
required eq('2.0'); onFalse setText('2.0'); onAbsence setText('2.0')
```

Poznámka: Za klíčovým slovem "`fixed`" může být zapsán kromě konstanty i odkaz na globální proměnnou se znakovým řetězcem:

```
<xd:declaration xd:scope="local"> String dnes = now().toString(); ... </xd:declaration>
...
   fixed dnes
```

V některých případech je vhodné s hodnotou fixed spojit i typ hodnoty. Např:

```
required float(); fixed '2.0'
```

3.2.12 Kontrola typu hodnot

Za specifikací výskytu může následovat popis validační akce, která je určena ke kontrole hodnoty atributu nebo textové hodnoty elementu. Výsledkem validační akce je logická hodnota "true" nebo "false". Pokud validační akce není popsána, je jejím výsledkem vždy "true", tj. každá hodnota, i prázdný řetězec. Pokud je však nastavena volba "ignoreEmptyAttributes" příslušný atribut je zcela ignorován.

Ke kontrole formátu běžně se vyskytujících hodnot je ve skriptu implementována řada in-line funkcí (). Kromě implementovaných kontrolních funkcí je možné popsat v X-definicích i vlastní typové funkce nebo použít externí funkce.

S kontrolou typu jsou svázány akce "onTrue" a případně "onFalse", které se provedou podle výsledku kontroly. Není-li uvedena žádná akce "onFalse" a je-li výsledkem validační funkce nepravda, provede se zápis o chybě do protokolového souboru, obsahujícího záznamy o zpracování.

Příklady kontroly typu a připojených akcí:

```
int(100,999); onTrue out(getText()); onFalse out('chyba');
string(10,20);
datetime('yyyyMMddHHmmss');
```

V případě, že nevystačíme s implementovanými kontrolními funkcemi, můžeme popsat uživatelskou kontrolní funkci, v deklarační sekci X-definice a odkázat se na ni jménem.

3.2.13 Implementované metody pro kontrolu typů

Základní kontrola typů zahrnuje typy odpovídající XML schématům. V tabulce 4a je seznam implementovaných metod. Tyto metody mohou obsahovat "pojmenované" parametry, odpovídající omezujícím podmínkám příslušných typů podle specifikace ve XML schématech.

V předposledním sloupci tabulky je zapsán typ výsledku parsované hodnoty. V posledním sloupci jsou popsány pojmenované a sekvenční parametry.

Povolené pojmenované parametry jsou uvedeny v následující tabulce a jsou popsány sekvencí odpovídajících písmen.

Tabulka 6 - Pojmenované parametry, které jsou v XML schématech

Pojmenovaný parametr odpovídající "facet" v XML schématech	Hodnota	Písmeno
%base	String se jménem base typu	b
%enumeration	Seznam povolených hodnot příslušného typu "[" ... "]"	e
%fractionDigits	Počet číslic zlomkové části čísla	f
%item	Sekcence s množinou parserů (Container)	i
%length	Délka řetězce, pole atd.	l
%maxExclusive	Hodnota příslušného typu, která stanoví, že výsledek parsování musí být menší	m
%maxInclusive	Horní hranice povolených hodnot	m
%maxLength	Maximální délka řetězce, pole atd.	l
%minExclusive	Hodnota příslušného typu, která stanoví, že výsledek parsování musí být větší	m
%minInclusive	Dolní hranice povolených hodnot	m
%minLength	Minimální délka řetězce, pole atd.	l
%pattern	Sekcence (Container) stringů s regulárními výrazy, které při zpracování musí být splněny	p
%totalDigits	Počet číslc celé části čísla	t
%whiteSpace	Způsob práce s mezerami v parsovaném textu. Možné hodnoty jsou: "collapse", "replace" nebo "preserve"	w

Např.:

`string(5, 10)` odpovídá zápisu `string(%minLength = 5, %maxLength = 10)`

`nebo decimal(3, 5)` odpovídá zápisu decimal(%fractionDigits=3, totalDigits=5)

Za sekvenčními parametry mohou být uvedeny pojmenované parametry:

```
string(5, 10, %whitespace = "preserve", %pattern = ["a*", "*.b"])
nebo decimal(3, 5, %minExclusive=-10, %maxExclusive=10)
```

Poznámka: metody, které zpracovávají datum kontrolují rok na hodnotu v rozsahu letošní rok -200 a letošní rok +200. Tato kontrola se může zrušit pomocí property "xdef.checkdate" na "false" (defaultní hodnota je "true"). Tedy datum 8.11.1620 je vyhodnoceno jako chyba pokud není nastaveno xdef.checkdate na false.

Seznam implementovaných kontrolních metod podle XML schémat je popsán v následující tabulce, kde jsou uvedena jednoduchá jména typů. Každý z těchto typů je možné zapsat také s prefixem "xs". Tedy např. string(5,10) je totéž jako xs:string(5,10). Jediný typ, u kterého je prefix "xs" povinný je typ xs:int, protože int bez prefixu z kompatibilních důvodů odpovídá typu XML schémat "long".

Podrobný popis datových typů z následující tabulky najdete v http://www.w3.org/TR/xmlschema11-2#datatype.

Tabulka 7 - Metody pro kontrolu typů shodných s typy v XML schématech

Jméno metody	Popis textové hodnoty	Typ výsledku	Parametry
anyUri	URI.	String	L belp
base64Binary	Pole bytů ve formátu base64.	Bytes	L belp
boolean	Boolean hodnota ("true", "false":).	Boolean	- p
byte	Hodnota byte (celé číslo).	Long	M bempt
date	Datum.	Datetime	M bemp
dateTime	Datum a čas.	Datetime	M bemp
decimal	Decimální číslo může mít desetinnou tečku.	Decimal	T befmpt
double	Číslo s pohyblivou řádovou čárkou.	double	M befmpt
duration	XML duration.	Duration	M bemp
ENTITY	Jméno XML entity.	String	L epl

ENTITIES	Seznam jmen XML entit oddělených mezerou.	Container	L epl
float	Číslo s pohyblivou řádovou čárkou.	double	M bemp
gDate	Datum.	Datetime	M bemp
gDay	Den datumu.	Datetime	M bemp
gMonth	Měsíc datumu.	Datetime	M bemp
gMonthDay	Měsíc a den datumu.	Datetime	M bemp
gYear	Rok datumu.	Datetime	M bemp
gYearMonth	Rok a měsíc datumu.	Datetime	M bemp
hexBinary	Pole bytů v hexadecimálním formátu.	Bytes	L belp
ID	Unikátní hodnota NCName v rámci zpracování XML dokumentu.	String	L belp
IDREF	Reference na unikátní hodnotu NCName v rámci zpracování XML dokumentu.	String	L belp
IDREFS	Seznam referencí na unikátní hodnoty NCName oddělených mezerami v rámci zpracování XML dokumentu.	Container	L belp
integer	Celé číslo.	long	M bempt
language	XML schema language.	String	L belp
list	Pole hodnot.	Container	L beilp
long	Celé číslo.	Long	M bempt
Name	Jméno (podle XML name).	String	L
NCName	Hodnota podle XML NCName.	String	L
negativeInteger	Celé záporné číslo.	Long	M bempt
NMTOKEN	XML NMTOKEN (tj. číslice, písmena, "_", "-", ".", ":").	String	L
NMTOKENS	Seznam XML NMTOKEN oddělených mezerou.	Container	L
nonNegativeInteger	Celé nezáporné číslo a nula.	Long	M bempt
nonPositiveInteger	Celé záporné číslo a nula. Pojmenovaný parametr %whiteSpace může mít hodnotu "replace" nebo "collapse". Default je "replace".	long	M belpw
normalizedString	Znakový řetězec.	String	L bempt
positiveInteger	Celé kladné číslo.	long	M bempt
QName	XML QName.	String	L belp
short	Celé číslo.	long	M bempt
string	Znakový řetězec. Pojmenovaný parametr %whiteSpace může mít hodnotu "replace", "collapse" nebo "preserve". Default je "preserve".	String	L belpw
time	Čas.	Datetime	M bemp
token	XML token (tj nesmí obsahovat uvnitř mezery).	String	L
union	Sjednocení typů.	Any	-, eip
unsignedByte	Hodnota byte bez znaménka.	long	M bempt
unsignedLong	Hodnota long bez znaménka.	Decimal	M bempt
unsignedInt	Hodnota integer bez znaménka.	long	M bempt
unsignedShort	Hodnota short bez znaménka.	long	M bempt
xs:int	Hodnota int.	long	M bempt

V X-definicích jsou definované některé další typy v následující tabulce.

Tabulka 8 - Metody pro kontrolu typů, které nejsou v XML schématech

Jméno metody	Popis textové hodnoty	Typ výsledku	Parametry
an	Alfanumerický řetězec (pouze písmena nebo číslice)	String	L
BNF(g, s)	Hodnota musí odpovídat pravidlu s z BNF gramatiky g	String	-
contains(s)	String který obsahuje s	String	-

containsi(s)	String který obsahuje s bez ohledu na velká/malá písmena	String	-
CHKID	Reference na unikátní token – podobně jako IDREF v tabulce 4a, ale výskyt referované hodnoty musí v tomto okamžiku existovat	String	-
CHKIDS	Sekvence referencí na unikátní tokeny – podobně jako IDREFS v tabulce 4a, ale výskyty referovaných hodnot musí v tomto okamžiku existovat	String	-
dateYMDhms	Datum a čas odpovídající masce "yyyyMMddHHmmss".	Datetime	M
dec	Desetinné číslo s desetinnou tečkou nebo čárkou (decimal v XML schéma (viz *Tabulka 7 - Metody pro kontrolu typů shodných s typy v XML schématech*), ale místo tečky může být i čárka)	Decimal	T efmpt
email	Emailová adresa	String	L
emailDate	Datum ve formátu pro email.	String	M
emailList	Seznam email adres oddělených čárkou nebo středníkem	Container	L
ends(s)	Hodnota musí končit hodnotou řetězce v parametru s.	String	-
endsi(s)	Hodnota musí končit hodnotou řetězce s bez ohledu na velká/malá písmena.	String	-
enum(s, s1, …)	Hodnota se musí shodovat s některým parametrem. Parametry s, s1, … musí byt stringy a musí to byt validni Java identifikatorty.	String	-
enumi(s, s1, …)	Hodnota se musí shodovat s některým parametrem bez ohledu na velka/mala pismena. Parametry s, s1, … musí byt stringy a musí to byt validni Java identifikatorty.	String	-
eq(s)	Hodnota se musí rovnat hodnotě řetězce s.	String	-
eqi(s)	Hodnota se musí rovnat hodnotě řetězce s bez ohledu na velká/malá písmena.	String	-
file	Hodnota musí být formálně správná souborová cesta	String	L
ISOdate	Datum podle ISO 8601 (parsuje i varianty, které nepodporuje date v XML schematech).	Datetime	M
ISOdateTime	Datum a čas podle ISO 8601	Datetime	M
ISOyear	Rok podle ISO 8601 (parsuje i varianty, které nepodporuje date v XML schematech).	Datetime	M
ISOyearMonth	Rok a měsíc podle ISO 8601 (parsuje i varianty, které nepodporuje date v XML schematech).	Datetime	M
languages	Seznam kódů jazyka podle ISO 639 nebo ISO 639-2 oddělených mezerou	Container	- elp
list(s1, s2, …)	Hodnota se musí rovnat některému z parametrů uvedených v seznamu parametrů.	String	-
listi(s1, s2, …)	Hodnota se musí rovnat některému z parametrů uvedených v seznamu parametrů bez ohledu na velká/malá písmena.	String	-
ListOf(t)	Hodnota se musí rovnat seznamu hodnot podle typu typové metody v parametru. Hodnoty jsou oddeleny bílými mezerami. DEPRECATED, nahraďte zápisem list(%item=t)	String	-
MD5	Hexadecimální reprezentace kontrolního součtu typu MD5 (32 hexadecimálních číslic)	Bytes	- e
NCNameList	Seznam NCName podle specifikace XML schéma. Oddělovač jsou mezery.	String	- elp
NCNameList(s)	Seznam NCName podle specifikace XML schéma. Seznam znaků sloužících jako oddělovač je v parametru s.	String	- elp
num	Hodnota je posloupnost číslic.	String	L
QNameList	Hodnota musí být seznam QName podle specifikace XML. Oddělovač jsou mezery.	Container	- elp
QNameList(s)	Hodnota musí být seznam QName podle specifikace XML. Seznam znaků sloužících jako oddělovač je v parametru s.	Container	- elp
QNameListURI	Hodnota musí být seznam kvalifikovaných jmen podle specifikace XML a pro každé jméno musí být definován namespace v kontextu	Container	- elp

	aktuálního elementu.		
QNameURI	Hodnota musí být Qname podle specifikace XML a musí mít definován namespace v kontextu aktuálního elementu	String	- elp
QNameURI (s)	Zkontroluje, zda existuje v aktuálním kontextu k zadanému kvalifikovanému jménu namespace URI odpovídající hodnotě v argumentu s. Jestliže argument je správné kvalifikované jméno a URI existuje, vrátí true, jinak vrátí false.	String	- elp
pic(s)	Hodnota musí odpovídat struktuře zobrazené znakovým řetězcem s, kde '9' znamená libovolnou číslici, 'A' libovolný alfabetický ASCII znak, 'X' libovolný alfanumerický (ASCII) znak a ostatní znaky se musí shodovat.	String	- elp
regex(s)	Hodnota musí odpovídat regulárnímu výrazu s. Řetězec s je interpretován jako zápis regulárního výrazu v jazyce Java.	RegexResult	-
sequence	Umožňuje popsat sekvenci různých hodnot. Parametrem %item=[typ1, typ2, …] můžeme popsat sekvenci parsovaných hodnot.	Container	L ielmp
SET	Uloží hodnotu do tabulky unikátních hodnot podobně jako ID, ale neohlásí chybu, pokud tato hodnota již existuje	String	-
starts(s)	Začátek hodnoty se musí rovnat hodnotě řetězce s.	String	-
startsi(s)	Začátek hodnoty se musí rovnat hodnotě řetězce s bez ohledu na velká/malá písmena.	String	-
tokens(s)	Hodnota se musí shodovat s některou částí masky s. Jednotlivé části masky jsou odděleny znakem "|".	String	-
tokensi(s)	Hodnota se musí shodovat s některou částí masky s. Jednotlivé části masky jsou odděleny znakem "|"bez ohledu na velká/malá písmena	String	-
uri	Hodnota musí být formálně správný zápis URI, jak je implementován v Java	String	-
uriList	Formálně správný seznam URI, jak je implementován v Java, oddělovač je čárka nebo whitespace	String	-
url	Hodnota musí být formálně správný zápis URL jak je implementován v Java	String	-
urlList	Hodnota musí být formálně správný seznam URL jak je implementován v Java, oddělovač je čárka nebo whitespace	String	-
xdatetime	Datum a/nebo čas odpovídající form8tu ISO 8601.	Datetime	-
xdatetime(s)	Datum a/nebo čas odpovídající masce s (viz *Tabulka 1 - Řídicí znaky v datumové masce* viz *Tabulka 1 - Řídicí znaky v datumové masce*).	Datetime	-
xdatetime(s,t)	Datum a/nebo čas odpovídající masce s (viz kapitola *Tabulka 1 - Řídicí znaky v datumové masce*). Výsledná hodnota bude přeformátována podle masky t.	Datetime	-

Z důvodů kompatibility se staršími verzemi jsou některá jména použita jako alias s jinými jmény typových metod (tj. vyvolá se stejná metoda, která je uvedena v některé z tabulek *Tabulka 8* nebo *Tabulka 9*).

V X-definicích jsou definované některé další typy v následující tabulce.

Tabulka 9 - Metody pro kontrolu typů, které mají z důvodů kompatibility alternativní jméno

Jméno metody	Popis textové hodnoty
base64	Viz base64Binary v *Tabulka 7 - Metody pro kontrolu typů shodných s typy v XML schématech* (alias)
Hex	Viz hexBinary v *Tabulka 7 - Metody pro kontrolu typů shodných s typy v XML schématech*: (alias)
ISOday	Viz gDay v *Tabulka 7 - Metody pro kontrolu typů shodných s typy v XML schématech* (deprecated)
ISOduration	Viz duration v *Tabulka 7 - Metody pro kontrolu typů shodných s typy v XML schématech* (deprecated)
ISOlanguage	Viz language v *Tabulka 7 - Metody pro kontrolu typů shodných s typy v XML schématech* (deprecated)
ISOlanguages	Viz languages v *Tabulka 8 - Metody pro kontrolu typů, které nejsou v XML schématech* (deprecated)
ISOmonth	Viz gMonth v *Tabulka 7 - Metody pro kontrolu typů shodných s typy v XML schématech* (deprecated)
ISOmonthDay	Viz gMonthDay v *Tabulka 7 - Metody pro kontrolu typů shodných s typy v XML schématech*

	(deprecated)
ISOtime	Viz time v *Tabulka 7 - Metody pro kontrolu typů shodných s typy v XML schématech* (deprecated)
NCname	Viz NCName v *Tabulka 7 - Metody pro kontrolu typů shodných s typy v XML schématech*(deprecated)
NCnameList(s)	Viz NCNameList(s) v *Tabulka 8 - Metody pro kontrolu typů, které nejsou v XML schématech* (deprecated)
normString	Viz normalizedString v *Tabulka 7 - Metody pro kontrolu typů shodných s typy v XML schématech* (deprecated)
normToken	Viz NMTOKEN v *Tabulka 7 - Metody pro kontrolu typů shodných s typy v XML schématech* (deprecated)
normTokens	Viz NMTOKENS v *Tabulka 7 - Metody pro kontrolu typů shodných s typy v XML schématech*(deprecated)
Qname	Viz QName v *Tabulka 7 - Metody pro kontrolu typů shodných s typy v XML schématech* (deprecated)
QnameList(s)	Viz QNameList(s) v *Tabulka 8 - Metody pro kontrolu typů, které nejsou v XML schématech* (deprecated)
QnameListURI	Viz QNameListURI(s) v *Tabulka 8 - Metody pro kontrolu typů, které nejsou v XML schématech* (deprecated)
QnameURI	Viz QNameURI v *Tabulka 8 - Metody pro kontrolu typů, které nejsou v XML schématech* (deprecated)
QnameURI (s)	Viz QNameURI(s) v *Tabulka 8 - Metody pro kontrolu typů, které nejsou v XML schématech* (deprecated)
rule(g, s)	Viz BNF v *Tabulka 8 - Metody pro kontrolu typů, které nejsou v XML schématech* (deprecated)

Pozn. Metody označené deprecated jsou zastaralé a není garantováno, že budou implementované v dalších verzích.

3.2.14 Deklarace typu

V některých případech je vhodné definovat vlastní typ hodnot. Deklarace typu se uvede klíčovým slovem "type" v sekci <xd:declaration>. Deklarovaný typ se chová podobně jako kontrolní metoda (nesmí však mít parametry). Příklad:

```
<xd:declaration>
   type mujtyp int(10, 20);
</xd:declaration>

<elem attr = "required mujtyp()"/>
```

Atribut "attr" musí vyhovovat kontrolní metodě typu int(10, 20).

Pokud uživatel nevystačí s uvedenými implementovanými typy, může definovat metodu pro vlastní typ. Výsledek může být typu boolean nebo ParseResult. Pokud je výsledek ParseResult, je možné navíc pracovat s parsovanou hodnotou. Příklad s ParseResult:

```
<xd:declaration xd:scope="global">
   ParseResult licheCislo() {
      ParseResult p = int();
      if (p.intValue() % 2 == 0) {
         p.error("Cislo musí byt liche!");
      }
      return p;
   }
</xd:declaration xd:scope="global">
<a a="licheCislo(); onTrue outln(getParsedValue() == 1)"/>
```

Vytiskne true, pokud atribut má hodnotu 1.

Příklad s boolean je jednodušší, ale v onTrue neumožňuje pracovat s parsovaným výsledkem:

```
<xd:declaration>
  boolean licheCislo {
     return parseInt(getText()) % 2 != 0 ? true : error("Cislo musí byt liche!");
   }
</xd:declaration>
<a a="required licheCislo(); onTrue outln(getParsedValue() == 1);"/>
```

Vytiskne vždycky false, protože metoda parsovaná hodnota je string a metoda getParsedValue v tomto případě nemá k dispozici zpracované číslo a tedy uvedená rovnost neplatí.

3.2.15 Tabulky unikátních hodnot a typové metody ID, SET, IDREF a CHKID

Pokud se vyžaduje vlastnost, že nějaká hodnota musí být v nějaké oblasti XML dokumentu jednoznačná, deklarujeme pro množinu těchto hodnot tabulku unikátních hodnot. Jednotlivé řádky tabulky jsou unikátní hodnoty (tj. nemohou

se v tabulce vyskytovat dvakrát). Implicitní součástí tabulky je objekt, který nazýváme klíč a popisuje řádky tabulky. Tento objekt může mít jednu nebo více položek, které se plní výsledky typových metod. K jednotlivým položkám klíče odpovídají typové metody, které je plní. Po naplnění požadovaných položek můžeme klíč uložit do tabulky metodou ID. Ta zkontroluje, zda klíč již v tabulce existuje a pokud ne, uloží do tabulky další řádek s jeho hodnotou. Pokud však existuje, ohlásí chybu, že klíč již v tabulce je. Naopak metodou IDREF se v požadované oblasti XML dokumentu zkontroluje, zda klíč v tabulce odkazuje na existující hodnotu v tabulce a pokud ne, ohlásí se chyba. V některých případech požadujeme, aby klíč v tabulce existoval již v okamžiku, kdy se na něj dotazujeme. To umožňuje metoda CHKID, která zjistí, zda klíč je v tabulce uložen v okamžiku volání metody. Metoda SET je podobná metodě ID, ale neohlásí chybu, pokud klíč v tabulce již existuje, tj. může se tento klíč opakovat. Metody ID, SET, IDREF, CHKID lze volat nad tabulkou, nebo nad kteroukoliv položkou klíče. V obou případech se použije aktuální hodnota klíče.

Tabulku unikátních hodnot deklarujeme zápisem "uniqueSet", za nímž následuje jméno tabulky a popis klíče. Klíč může mít jednu nebo více položek. Každá položka má jméno a dále musí být uvedena odpovídající metoda kontroly typu. Např.:

```
uniqueSet tab {prvni: int; druhy: string(5, 10)}
```

Někdy můžeme požadovat, aby určitá položka klíče nebyla naplněna, tj. měla hodnotu null (i to je samozřejmě hodnota). V takovém případě před popis typu položky zapíšeme znak "?":

```
uniqueSet tab {prvni: int; druhy: ? string(5, 10)}
```

Otazník před typem má ještě další význam. Metody ID, IDREF a CHKID použijí hodnotu klíče a ponechají hodnotu klíče nezměněnou. Avšak, je-li typ položky deklarován s otazníkem, dosadí se po uplatnění klíče (tj. po zavolání některé z metod ID, IDREF, CHKID) do této položky hodnota null.

Typová kontrola se vyvolá, zapíšeme-li jméno tabulky, dále znak "." a jméno položky, viz následující příklad:

```
<xd:declaration xd:scope="local">
  uniqueSet dum{cislo: int; byt: ?int; osoba: ? string; firma: ? string}
</xd:declaration>
<Ulice>
  <Dum xd:script="+" Cislo="dum.cislo.SET() /*do tabulky dum se ulozi klic s polozkou cislo*/ ">
    <Najemnik xd:script="*" jmeno="dum.osoba.ID() /*ulozi se klic s polozkami cislo, osoba */" />
    <Firma xd:script="*" jmeno="dum.firma.ID() /*ulozi se klic s polozkami cislo, firma */" />
  </Dum>
  <Najemnik xd:script="*; finally dum.IDREF() /*polozky maji hodnotu cislo a osoba nebo firma */"
    Dum="dum.cislo"
    Osoba="? dum.osoba()"
    Firma="? dum.firma()" />
</Ulice>
```

V kterémkoliv okamžiku můžeme nad tabulkou zavolat metodu CLEAR. Tato metoda zkontroluje, zda v tabulce existují odkazy na neexistující klíče a pokud ano, ohlásí příslušné chyby a posléze všechny řádky tabulky vymaže. Pokud tuto metodu nezavoláme, vyvolá se automaticky před ukončením platnosti objektu s tabulkou (je-li tabulka deklarovaná v sekci <xd:declaration>, pak se tato metoda vyvolá na konci zpracování XML dokumentu).

Chceme-li v nějakém okamžiku vynulovat všechny položky klíče, můžeme zavolat metodu NEWKEY. Ta nemá vliv na obsah tabulky, ale dosadí null do všech položek klíče.

3.2.16 Tabulka unikátních hodnot bez pojmenovaných položek

V sekci typové kontroly je možné stejné, jako v XML schématech zapsat ID() resp. IDREF(). K tomu účelu je vygenerována jedna globální tabulka, která má v řádcích hodnoty string, který je parsován jako NCName. Uživatel si podobnou tabulku může deklarovat a navíc stanovit jakýkoliv typ hodnoty. Takovou tabulku může deklarovat příkazem:

```
<xd:declaration>
  uniqueSet cislo: int;
</xd:declaration xd:scope="local">

<Domy>
  /*do tabulky dum se uloží klíč s číslem*/
  <Dum    xd:script="+" Cislo="cislo.ID()/>

  /*číslo musí již v tabulce být*/
  <Cislo  xd:script="*" Cislo="cislo.IDREF()"/>
</Domy>
```

3.2.17 Propojení tabulek unikátních hodnot

Tabulky v některých případech potřebujeme propojit. To docílíme tak, že jako parametr typové metody s klíčem zapíšeme typovou hodnotu s jinou tabulkou. Upozorněme, že odkaz na položku jiné tabulky znamená jen převzetí odpovídající typové metody. Použití je zřejmé z následujícího příkladu:

```
<xd:declaration xd:scope="local">
  uniqueSet dum{cislo: int; byt: ? int; osoba: ? string; firma: ? string}
  uniqueSet ulice{nazevUlice: string; cisloDomu: dum.cislo;}
</xd:declaration>

<Mesto Nazev="string()">
  <Ulice xd:script="+" Nazev="ulice.nazevUlice()">
    <Dum xd:script="+" Cislo="dum.cislo(ulice.cisloDomu.ID())" />
      <Najemnik xd:script="*" jmeno="dum.osoba.ID()" />
      <Firma    xd:script="*" jmeno="dum.firma.ID()" />
    </Dum>
  </Ulice>
</Mesto>
```

Při zpracování atributu Nazev v elementu <Ulice> se nastaví položka klíče "ulice.nazevUlice". Při zpracování atributu Cislo v elementu <Dum> se nastaví položka klíče "ulice.cisloDomu", tento klíč (tj. dvojice (nazevUlice, cisloDomu)) se uloží do tabulky ulice, zkontroluje se jeho unikátnost a nastaví se položka klíče "dum.cislo".

3.2.18 Specifikace výskytu elementů

U popisu vnitřních elementů modelu elementu se uvádí pouze specifikace výskytu (specifikace kontroly typu nemá smysl). Specifikace výskytu elementu může být zapsána jednou z následujících forem:

"?"	-	element se nemusí vyskytnout nebo se může vyskytnout jednou (totéž jako “optional” nebo 0..1)
"*"	-	element se nemusí vyskytnout nebo počet výskytů není omezen (totéž jako “optional” nebo 0..*)
"+"	-	element se musí vyskytnout jednou nebo vícekrát (totéž jako “required” nebo 1..*)
"m"	-	element se musí vyskytnout právě m-krát
"m..n"	-	element se musí vyskytnout m-krát a smí se vyskytnout až n-krát
"n..*"	-	element se musí vyskytnout n-krát a smí se vyskytnout neomezeně
"required"	-	element se musí vyskytnout právě jednou (totéž jako 1 nebo 1..1)
"optional"	-	element se může vyskytnout maximálně jednou (totéž jako 0..1)
"ignore"	-	element se sice může vyskytnout, ale jeho výskyt je v dalším zpracování ignorován
"illegal"	-	element se nesmí vyskytnout.

Ke specifikaci výskytu se vztahují události "onExcess" a "onAbsence". Akce "onExcess" se provede, jestliže v okamžiku výskytu daného objektu je překročena maximální hranice výskytu. Akce "onAbsence" se provede, jestliže nebyl dosažen minimální předepsaný počet výskytů.

Poznámka: z důvodů kompatibility je možno před zápis výskytu zapsat klíčové slovo „occurs“. Specifikace „required“ je defaultní a je možné ji vynechat.

3.2.19 Použití šablony (Template element)

Někdy je vhodné popsat element "jedna k jedné", jinými slovy všechny hodnoty atributů, textových hodnot či potomků včetně jejich výskytu jsou konstantní. V režimu konstrukce pak takový element je zkopírován do výsledku. Toho lze docílit zapsáním slova "template" do skriptu modelu elementu. Nicméně, začíná-li na nějakém místě textová hodnota znaky "$$$script:", zpracuje se příslušná hodnota jako zápis skriptu:

```
<elem xd:script = "template"
      attr1 = "abcd"
      attr2 = "$$$script: optional datetime('yyyy/M/d'); create now().toString('yyyy/M/d')">
   <child1/>
   <child2>text1</child2>
   <child2>text2</child2>
</elem>
```

Výsledek je ekvivalentní zápisu:

```
<elem xd:script = " occurs 1; create newElement()"
    attr1 = "fixed 'abcd'; create newElement()"
    attr2 = "optional datetime('yyyy/M/d'); create now().toString('yyyy/M/d')">
```

```
  <child1 xd:script = "occurs 1; create newElement()"/>
  <child2 xd:script = "occurs 1; create newElement()">
     fixed 'text1'; create newElement()
   </child2>
  <child2 xd:script = "occurs 1; create newElement()">
     fixed 'text2'; create newElement()
   </child2>
</elem>
```

Poznámka: Za klíčové slovo "template" je možné za středníkem doplnit "options trimText" nebo "options noTrimText". Není-li nastavena volba trimText, považují se všechny mezery, nové řádky a tabelátory mezi elementy za povinné literály (tj. textové konstanty)!

3.2.20 Příkazy skriptu

Příkazem skriptu může být buď volání metody ukončené středníkem, nebo složený příkaz, podobný tělu metody v jazycích "C" nebo "java". Výkonné příkazy jsou pak umístěné mezi znaky "{" a "}" a návrat je možné provést příkazem "return" obdobně jako v metodě. Výrazy a výkonné příkazy jsou podobné jako v jazycích "java" nebo "C" (včetně složeného příkazu).

Syntaxe příkazů je shodná s jazykem "Java". Implementované příkazy jsou:

- příkaz deklarace proměnné (typy hodnot ovšem musí odpovídat typům ve skriptu)
- přiřazovací příkaz
- příkaz volání metody
- příkaz "break"
- příkaz "continue"
- příkaz "do"
- příkaz "for"
- příkaz "if"
- příkaz "return"
- příkaz "switch"
- příkaz "throw"
- příkaz "try" a "catch"
- příkaz "while"

Podrobný popis by překročil rámec tohoto textu. Čtenář může najít popis např. v [5][6] .

3.2.21 Implementované metody skriptu

V kódu příkazů skriptu je možné volat řadu implementovaných metod a funkcí, které je možné použít na různých místech skriptu. V následujících tabulkách jsou uvedeny typy parametrů metod písmeny, které označují typ parametru:

AnyValue	v, v1, v2, …
Datetime	d
Element	e, e1, e2, …
Container	c, c1, c2, …
int:	n, n1, n2, …
float	f, f1, f2, …
Object	o
String	s, s1, s2, …

Tabulka 10 - Základní metody

Jméno metody	Popis	Výsledek	Použití
addComment(s)	Přidá komentářový uzel s hodnotou s k potomkům aktuálního elementu.		Element, attribut, text
addText(s)	Přidá textový uzel s hodnotou s k potomkům aktuálního elementu.		Element, attribut, text

clearReports()	vymaže všechna aktuální (temporální) hlášení o chybách, generovaná předcházejícím příkazem. (používá se v akci onFalse).		všechny akce
cancel()	Násilné ukončení procesoru X-definic		všechny akce
cancel(s)	Násilné ukončení procesoru X-definic se zprávou o ukončení s		všechny akce
compilePattern(s)	Přeloží regulární výraz s. Deprecated, nahrazeno new Regex.	Regex	všechny akce
defaultError()	Zapíše do pracovního protokolu chybu, kterou nastavila předchozí metoda pro testovaní hodnoty a vrátí hodnotu "false". Do protokolu se zapíše chyba XDEF515 (Hodnota se liší od očekávané).	boolean	všechny akce
easterMonday(n)	Vrátí datum s velikonočním pondělím pro rok n	Datetime	všechny akce
error(r)	Zapíše se chybové hlášení podle Reportu r do dočasného souboru s protokolem. Výsledkem funkce je hodnota "false".	boolean (vždy "false")	všechny akce
error(s)	Zapíše se chybové hlášení s do dočasného souboru s protokolem. Výsledkem funkce je hodnota "false".	boolean (vždy "false")	všechny akce
error(s1,s2)	Zapíše se chybové hlášení do dočasného souboru s protokolem. Výsledkem funkce je hodnota "false". Parametr s1 je identifikátor chyby, parametr s2 je text chyby.	boolean (vždy "false")	všechny akce
error(s1,s2, s3)	Zapíše se chybové hlášení do dočasného souboru s protokolem. Výsledkem funkce je hodnota "false". Parametr s1 je identifikátor chyby, parametr s2 je text chyby, parametr s3 je modifikátor s parametry textu chyby.	boolean (vždy "false")	všechny akce
errors()	Vrátí aktuální počet chyb hlášených během zpracování.	int	všechny akce
errorWarnings()	Vrátí aktuální počet chyb a varování hlášených během zpracování.	int	všechny akce
format(s, v1, ...)	Vrací řetězec vytvořený z hodnot parametrů v1, ... podle masky s (viz metoda format v java.lang.String).	String	všechny akce
format(l, s, v1, ...)	Vrací řetězec vytvořený z hodnot parametrů v1, ... podle oblasti od hodnoty l (Locale) a masky s (viz metoda format v java.lang.String)..	String	všechny akce
from()	Vrátí Container odpovídající aktuálnímu kontextu. Metodu je možné použít pouze v akci "create" ve skriptu elementu, textové hodnoty nebo atributu. Je-li výsledek null, vrátí se prázdný Container.	Container	create element
from(s)	Vrátí Container po provedení výrazu xpath za stringu s na aktuálním kontextu. Metodu je možné použít pouze v akci "create" ve skriptu elementu, textové hodnoty nebo atributu. Je-li výsledek null, vrátí se prázdný Container.	Container	create attribut create element create text
from(e, s)	Vrátí Container po provedení výrazu xpath na elementu e. Metodu je možné použít pouze v akci "create" ve skriptu elementu, textové hodnoty nebo atributu. Je-li e null, je výsledek prázdný Container	Container	create attribut create element create text
fromXQ(s)	Vrátí Container po provedení výrazu xquery ze stringu s na aktuálním kontextu. Metodu je možné použít pouze v akci "create" ve skriptu elementu, textové hodnoty nebo atributu. Je-li výsledek null, vrátí se prázdný Container. **UPOZORNENI:** tato metoda je implementována pouze pro verzi s knihovnou XPath2 a Saxon.	Container	create attribut create element create text
fromXQ(e, s)	Vrátí Container po provedení výrazu xquery na elementu e. Metodu je možné použít pouze v akci "create" ve skriptu	Container	create attribut create element

	elementu, textové hodnoty nebo atributu. Je-li e null, je výsledek prázdný Container. **UPOZORNENI:** tato metoda je implementována pouze pro verzi s knihovnou XPath2 a Saxon.		create text
getAttr(s)	Vrátí hodnotu atributu, se jménem s (z aktuálního elementu). Pokud tento atribut neexistuje, vrátí prázdný string.	string	Element
getAttr(s1,s2)	Vrátí hodnotu atributu, s lokálním jménem s1 a namespace s2 (z aktuálního elementu). Pokud tento atribut neexistuje, vrátí prázdný string.	string	Element
getAttrName()	Jméno aktuálního atributu.	String	všechny akce ve skriptu atributů
getElement()	Výsledkem je aktuální element.	Element	Element, text atribut
getElementName()	Jméno aktuálního elementu.	String	všechny akce
getElementLocalName()	Lokální jméno aktuálního elementu.	String	všechny akce
getElementText()	Vrátí string se spojeným obsahem textovývh uzlů, které jsou přímí potomci aktuáílního elementu.	String	všechny akce
getImplProperty(s)	Vrátí hodnotu property z aktuální X-definice, která se jmenuje s. Pokud příslušná položka neexistuje vrátí prázdný string.	String	všechny akce
getImplProperty(s1,s2)	Vrátí hodnotu property z X-definice s1, která se jmenuje s2. Pokud příslušná X-definice nebo položka neexistuje vrátí prázdný string.	String	všechny akce
getIem(s)	Vrátí hodnotu položky z aktuálního kontextu.	String	Element, text atribut
getLastError()	Vrátí poslední hlášenou chybu.	Message	Element, text atribut
getMaxYear()	Vrátíí maximální povolenou hodnotu roku při parsování datumu.	int	všechny akce
getMinYear()	Vrátíí minimální povolenou hodnotu roku při parsování datumu.	int	všechny akce
getNamespaceURI()	Vrátí řetězec, jehož hodnota je namespace URI aktuálního elementu. Jestliže namespace neexistuje vrátí prázdný řetězec.	String	všechny akce
getNamespaceURI (n)	Vrátí řetězec, jehož hodnota je namespace URI uzlu n (může být element nebo atribut). Jestliže namespace neexistuje vrátí prázdný řetězec.	String	všechny akce
getNamespaceURI (s)	Vrátí řetězec, jehož hodnota je namespace URI odpovídající prefixu s v kontextu aktuálního elementu. Jestliže neexistuje vrátí prázdný řetězec.	String	všechny akce
getNamespaceURI (s,e)	Vrátí řetězec, jehož hodnota je namespace URI odpovídající prefixu s v kontextu elementu e. Jestliže neexistuje vrátí prázdný řetězec.	String	všechny akce
getOccurrence()	Vrátí aktuální počet výskytů uzlu v aktuáním elementu.	int	Element
getParentContextElement()	Vrátí element kontextu rodiče právě zpracovávaného elementu	Element	Element
getParentContextElement(n)	Vrátí element kontextu n-tého rodiče právě zpracovávaného elementu	Element	Element
getParsedBoolean()	Vrátí objekt s hodnotou boolean (pokud v předchozím skriptu byla přečtena typovou metodou).	boolean	Pouze po metodě typové kontroly

	Upozornění: metoda musí být volána bezprostředně po volání typové metody a jen prošla typová kontrola, tj., např. v sekci onTrue. Pokud není tato podmínka splněna, pak není výsledek metody definován.		
getParsedBytes()	Vrátí objekt s hodnotou pole bytů (pokud v předchozím skriptu bylo přečteno typovou metodou). *Upozornění: metoda musí být volána bezprostředně po volání typové metody a jen prošla typová kontrola, tj., např. v sekci onTrue. Pokud není tato podmínka splněna, pak není výsledek metody definován.*	Bytes	Pouze po metodě typové kontroly
getParsedDatetime()	Vrátí objekt s datem (pokud v předchozím skriptu bylo přečteno buď typovou metodou). *Upozornění: metoda musí být volána bezprostředně po volání typové metody a jen prošla typová kontrola, tj., např. v sekci onTrue. Pokud není tato podmínka splněna, pak není výsledek metody definován.*	Datetime	Pouze po metodě typové kontroly
getParsedDecimal()	Vrátí objekt s dekadickým číslem (pokud v předchozím skriptu bylo přečteno typovou metodou). *Upozornění: metoda musí být volána bezprostředně po volání typové metody a jen prošla typová kontrola, tj. např. v sekci onTrue. Pokud není tato podmínka splněna, pak není výsledek metody definován.*	Decimal	Pouze po metodě typové kontroly
getParsedDuration()	Vrátí objekt s hodnotou časového intervalu (pokud v předchozím skriptu byla přečtena typovou metodou). *Upozornění: metoda musí být volána bezprostředně po volání typové metody a jen prošla typová kontrola, tj. např. v sekci onTrue. Pokud není tato podmínka splněna, pak není výsledek metody definován.*	Duration	Pouze po metodě typové kontroly
getParsedFloat()	Vrátí objekt s hodnotou float (pokud v předchozím skriptu byla přečtena typovou metodou). *Upozornění: metoda musí být volána bezprostředně po volání typové metody a jen prošla typová kontrola, tj. např. v sekci onTrue. Pokud není tato podmínka splněna, pak není výsledek metody definován.*	Float	Pouze po metodě typové kontroly
getParsedInt()	Vrátí objekt s hodnotou int (pokud v předchozím skriptu byla přečtena typovou metodou). *Upozornění: metoda musí být volána bezprostředně po volání typové metody a jen prošla typová kontrola, tj. např. v sekci onTrue. Pokud není tato podmínka splněna, pak není výsledek metody definován.*	int	Pouze po metodě typové kontroly
getParsedValue()	Vrátí objekt s hodnotou, která v předchozím skriptu byla přečtena typovou metodou. *Upozornění: metoda musí být volána bezprostředně po volání typové metody a jen prošla typová kontrola, tj. např. v sekci onTrue. Pokud není tato podmínka splněna, pak není výsledek metody definován.*	AnyValue	Pouze po metodě typové kontroly
getQnameLocalpart(s)	Vrátí řetězec s lokálním jménem argumentu, který je qname.	String	všechny akce
getQnamePrefix(s)	Vrátí řetězec s prefixem argumentu, který je qname. Vrátí prázdný řetězec, pokud argument není qname nebo nemá prefix.	String	všechny akce
getQnameURI(s)	Vrátí řetězec, jehož hodnota je namespace URI odpovídající qname s v kontextu aktuálního elementu. Jestliže namespace neexistuje vrátí prázdný řetězec.	String	všechny akce
getQnameURI(s, e)	Vrátí řetězec, jehož hodnota je namespace URI odpovídající qname s v kontextu elementu e. Jestliže namespace	String	všechny akce

	neexistuje vrátí prázdný řetězec.		
getRootElement()	Vrátí root element z aktuálního elementu.	Element	element, attribut, text
getSpecialDates()	Vrátí Container s hodnotami datumu, které jsou povoleny i když rok datumu není v povoleném rozsahu.	Container	všechny akce
getText()	Vrátí string s hodnotou atributu nebo textového uzlu.	String	atribut, text
getXDPosition()	Vrátí string s aktuální X-pozicí X-definice.	String	element, atribut, text
getXpos()	Vrátí aktuální pozici zpracovávaného XML dokumentu ve formátu XPath.	String	element, attribut, text
getUserObject()	Vrátí externí objekt.	Object	všechny akce
getVersionInfo()	Vrátí informaci o použité verzi X-definic..	String	všechny akce
hasAttr(s)	Vrátí true když v aktuálním elementu je atribut s.	boolean	Element
hasAttr(s1, s2)	Vrátí true, jestliže v aktuálním elementu je atribut s1 lokálním jménem s a namespace s2.	boolean	Element
IsCreateMode()	Vrátí true, jestliže zpracování probíhá v konstrukčním režimu.	boolean	všechny akce
isDatetime(s)	Výsledek je true, když datum ve stringu s odpovídá formátu podle normy ISO 8601 (tj. masce "`y-M-d[TH:m:s[.S][Z]]`").	boolean	všechny akce
isDatetime(s1, s2)	Výsledek je true, když datum ve stringu s1 odpovídá formátu v masce s2.	boolean	všechny akce
isLeapYear(n)	Vrátí true, je-li rok n přestupný.	boolean	všechny akce
isNumeric(s)	Vrátí true, když string s obsahuje pouze číslice.	boolean	všechny akce
newElement()	Vytvoří nový element (jméno se odvodí podle místa, kde byla metoda specifikována).	Element	Create
newElement(s)	Vytvoří nový element pojmenovaný podle argumentu s.	Element	všechny akce
newElement(s1, s2)	Vytvoří nový element pojmenovaný podle argumentu s1 a majici napespace s2.	Element	všechny akce
newElements(n)	Vytvoří Container s n novými elementy (jméno se odvodí podle místa, kde byla metoda specifikována).	Container	Create
newElements(n,s)	Vytvoří Container s n novými elementy pojmenovanými podle argumentu s.	Container	všechny akce
newElements(n,s1,s2)	Vytvoří Container s n novými elementy pojmenovanými podle argumentu s1 a namespace s2.	Container	všechny akce
now()	výsledkem je dnešní datum a okamžitá hodnota času	Datetime	všechny akce
occurrence()	Vrátí číslo, odpovídající aktuálnímu počtu výskytu elementu.	int	Element, text
out(v)	hodnota v převedena na řetězec a je uložena na standardní výstup		všechny akce
outln()	Výstup znaku "nová řádka" do standardního výstupního souboru		všechny akce
outln(v)	hodnota v převedena na řetězec a pripojí se znak "nová řádka" a zapíše se do standardního výstupního souboru		všechny akce
parseBase64(s)	Převede string s do pole bytů. Jestliže při parsování dojde k chybě, vrátí hodnotu null.	Bytes	všechny akce
parseDate(s)	Převede string s s datumem ve formátu ISO 8601 na datum (tj. podle masky "`yyyy-M-dTH:m[:s][.S][Z]`"). Jestliže při parsování dojde k chybě, vrátí hodnotu null.	Datetime	všechny akce
parseDate(s1,s2)	Převede string s1 na datum podle masky v parametru s2. Jestliže při parsování dojde k chybě, vrátí hodnotu null.	Datetime	všechny akce

parseEmailDate(s)	Převede string s s datumem ve formátu RFC822 na datum. Jestliže při parsování dojde k chybě, vrátí hodnotu null.	Datetime	všechny akce
parseFloat(s)	Převede string s na hodnotu float. Jestliže při parsování dojde k chybě, vrátí hodnotu null.	float	všechny akce
parseInt (s)	Převede string s na hodnotu integer. Jestliže při parsování dojde k chybě, vrátí hodnotu null.	int	všechny akce
parseDuration(s)	Převede string s časovým intervalem ve formátu ISO 8601 na Duration. Jestliže při parsování dojde k chybě, vrátí hodnotu null.	Duration	všechny akce
parseHex(s)	Převede string s do pole bytů. Jestliže při parsování dojde k chybě, vrátí hodnotu null.	Bytes	všechny akce
pause()	V režimu "debug" vytiskne na standardní výstup informační řádku. Program se zastaví a čeká na odpověď na standardním vstupu. Je-li odpověď "go", program pokračuje. Kromě "go" je možné zadat další příkazy, jejichž seznam se vytiskne zadáním "?" nebo nesprávného příkazu. Pokud není nastaven režim "debug" je tato metoda ignorována.		všechny akce
pause(s)	V režimu "debug" vytiskne na standardní výstup informační řádku a text s. Program se zastaví a čeká na odpověď na standardním vstupu. Je-li odpověď "go", program pokračuje. Kromě "go" je možné zadat další příkazy, jejichž seznam se vytiskne zadáním "?" nebo nesprávného příkazu.Pokud není nastaven režim "debug" je tato metoda ignorována.		všechny akce
printf(s, v1, ...)	vytiskne do standardního výstupnu řetězec vytvořený z hodnot parametrů v1, ... podle masky s (viz metoda printf v java.io.PrintStream).		všechny akce
printf(l, s, v1, ...)	vytiskne do standardního výstupu řetězec vytvořený z hodnot parametrů v1, ... podle oblasti c larametru l (Locale) a masky s (viz metoda printf v java.io.PrintStream).		všechny akce
removeAttr(s)	Odstraní atribut se jménem s z aktuálního elementu.		Element, atribut, text
removeAttr(s1, s2)	Odstraní atribut s lokálním jménem s1 a namespace s2 z aktuálního elementu.		Element, atribut, text
removeText()	Odstraní aktuální (právě zpracovávaný) uzel s textovou hodnotou (tj. atribut nebo textový uzel)		atribut, text
removeWhiteSpaces(s)	Všechny výskyty sekvencí mezer, tabelátorů a nových řádek ve stringu s se nahradí jedinou mezerou.	String	všechny akce
replace(s1, s2, s3)	Všechny výskyty řetězce s2 ve stringu s1 se nahradí řetězcem s3.	String	všechny akce
replaceFirst(s1, s2, s3)	První výskyt řetězce s2 v řetězci s1 se nahradí řetězcem s3.	String	všechny akce
returnElement(o)	Vrátí jako výsledek parsování element vyrobený z argumentu a ukončí procesor X-definic.		
setAttr(s1,s2)	Nastaví v právě zpracovávaném elementu atribut se jménem s1 na hodnotu s2.		Element, atribut, text
setAttr(s1,s2,s3)	Nastaví v právě zpracovávaném elementu atribut s lokálním jménem s1 a s namespace s2 na hodnotu s3.		Element, atribut, text
setElement (e)	Vloží na aktuální místo zpracovávaného XML element e (např. se takto může doplnit element v sekci onAbsence)		Element, atribut, text
setMaxYear(n)	Nastaví maximální hodnotu roku při parsování datumu.		všechny akce
setMinYear(n)	Nastaví minimální hodnotu roku při parsování datumu.		všechny akce
setParsedValue(v)	Uloží hodnotu v do objektu po parsování. *Upozornění: metoda musí být volána bezprostředně po*	ParsedValue	Pouze po metodě typové kontroly

	volání typové metody a jen prošla typová kontrola, tj. např. v sekci onTrue. Pokud není tato podmínka splněna, pak není výsledek metody definován.		
setSpecialDates(c)	Nastaví hodnoty datumu, které jsou povoleny i když rok datumu není v povoleném rozsahu.		všechny akce
setText(s)	Řetězec, do kterého je převedena hodnota s v nahradí aktuální hodnotu atributu nebo textového uzlu		text, atribut
setUserObject(o)	Nastaví externí Java objekt.		všechny akce
tail(s,n)	Vrátí posledních n znaků ve stringu s	String	všechny akce
toString(x)	Převede hodnotu x standardním způsobem na řetězec znaků. Výraz v může být typu String, Integer, Float, Date, Element a List	String	všechny akce
toString(x, s)	Převede hodnotu x na řetězec znaků. Výraz v může být typu Integer, Float nebo Date. Hodnota s je maska formátu. Maska byla již popsána u jednotlivých typů v kapitole 103.2.3.		všechny akce
trace()	V režimu "debug" vytiskne na standardní výstup informační řádku. Pokud není nastaven režim "debug" je tato metoda ignorována.		všechny akce
trace(s)	V režimu "debug" vytiskne na standardní výstup informační řádku a text s. Pokud není nastaven režim "debug" je tato metoda ignorována.		všechny akce
translate(s1,s2,s3)	Nahradí v řetězci s1 všechny výskyty znaků, které odpovídají některému znaku v řetězci s2 znakem na odpovídající pozici v řetězci s3. Např: translate("bcr","abc","ABa") vrátí "Bar". Pokud není na příslušné pozici v řetězci s2 žádný znak, pak se tento znak vynechá: `translate("-abc-","ab-","BA")` vrátí `"BAc"`	String	všechny akce
xcreate(c)	výsledkem je objekt, sestavený podle modelu elementu c (používá se v režimu konstrukce).	Element	všechny akce
xparse(s)	Parsuje XML dokument ze stringu s podle aktuální X-definice a vrátí root element. Je-li s string, popisující URL nebo cestu k souboru, parsing probíhá ze streamu, začíná-li string znakem "<", parsuje se přímo hodnota stringu.	Element	všechny akce
xparse(s1, s2)	Parsuje XML dokument ze stringu s1 podle X-definice se jménem s2 a vrátí root element. Je-li s1 string, popisující URL nebo cestu k souboru, parsing probíhá ze streamu, začíná-li string znakem "<", parsuje se přímo hodnota stringu.	Element	všechny akce
xparse(s1, $null)	Parsuje XML dokument ze stringu s1 bez X-definice. Vrátí root element. Je-li s string, popisující URL nebo cestu k souboru, parsing probíhá ze streamu, začíná-li string znakem "<", parsuje se přímo hodnota stringu.	Element	všechny akce
xpath(s)	Vrátí Container po provedení výrazu xpath zadaného stringem s na aktuálním elementu. Je-li akruální element null, vrátí prázdný Container.	Container	element, atribut, text
xpath(s, c)	Vrátí Container po provedení výrazu xpath zadaného stringem s na elementu (elementech) z Containeru c. Je-li výsledek null, vrátí prázdný Container.	Container	všechny akce
xquery(s)	Vrátí Container po provedení výrazu xquery ze stringu s na aktuálním elementu. Metodu je možné použít pouze v sekkci "create" skriptu elementu, textové hodnoty nebo atributu. Je-li aktuální element null, vrátí prázdný Container	Container	element, atribut, text

	UPOZORNENI: tato metoda je implementována pouze pro verzi s knihovnou XPath2 a Saxon.		
xquery(s,e)	Vrátí Container po provedení výrazu xquery ze stringu s na elementu e. Je-li aktuální element null, vrátí prázdný Container. **UPOZORNENI:** tato metoda je implementována pouze pro verzi s knihovnou XPath2 a Saxon.	Container	všechny akce

V následujících tabulkách popíšeme metody, které jsou implementované nad jednotlivými typy objektů. Typu objektu odpovídá jméno (např. "int", "String" atd. – viz odstavec 3.2.3). Rovněž je každému typu přiřazeno číslo (tzv. Typové ID). Jména typů a odpovídající identifikátory ve skriptu a v Java kódu podle interface cz.syntea.xdef.XDValueTypes jsou v následující tabulce).

Tabulka 11 - Jména typů skriptu a odpovídající ID typů

Jméno typu ve skriptu	ID typu skriptu	Jméno položky ve třídě XDValueType
Boolean	$BOOLEAN	BOOLEAN
BNFGrammar	$BNFGRAMMAR	BNFGRAMMAR
BNFRule	$BNFRULE	BNFRULE
Bytes	$BYTES	BYTES
Container	$CONTAINER	CONTAINER
Datetime	$DATETIME	DATETIME
Decimal	$DECIMAL	DECIMAL
Duration	$DURATION	DURATION
Element	$ELEMENT	ELEMENT
Exception	$EXCPTION	EXCEPTION
Float	$FLOAT	DOUBLE
Input	$INSTREAM	INSTREAM
Int	$INT	LONG
NamedValue	$NAMEDVALUE	NAMED
Object	$OBJECT	OBJECT
Output	$OUTSTREAM	OUTSTREAM
Parser	$PARSER	PARSER
ParseResult	$PARSERESULT	PARSERESULT
Regex	$REGEX	REGEX
RegexResult	$REGEXRESULT	REGEXRESULT
Report	$REPORT	REPORT
ResultSet	$RESULTSET	RESULTSET
Service	$SERVICE	SERVICE
Statement	$STATEMENT	STATEMENT
String	$STRING	STRING
XModel	$XMODEL	XMODEL
XmlOutStream	$XMLWRITER	XMLWRITER
XPathExpr	$XPATH	XPATH_EXPR
XQueryExpr	$XQUERY	XQUERY_EXPR

Pro každý typ uvedený v předchozí tabulce jsou implementovány následně uvedené metody:

Tabulka 12 - Metody nad objekty všech typů

Jméno metody	Popis	Výsledek

x.toString()	Vrátí string v "zobrazitelném" tvaru hodnoty x.	String
typeName(v)	Vrátí jméno typu v.	String
valueType(v)	Vrátí ID typu	Int

Tabulka 13 - Metody nad objekty typu BNFGrammar

Jméno metody	Popis	Výsledek
BNFGrammar x	Konstuktor BN gramatiky x,se zapisuje do elementu <xd:BNFGrammar name="x"> Text BNF gramatiky … viz 5.1	BNFGrammar
BNFGrammar x	Konstuktor BN gramatiky x, která je rozšíření gramatiky g,se zapisuje do elementu <xd:BNFGrammar name="x" extends="g"> Text BNF gramatiky … viz 5.1	BNFGrammar
x.parse(s)	Vrátí zparsovanou hodnotu textové hodnoty nebo atributu podle pravidla s gramatiky x	ParseResult
x.parse(s1, s2)	Vrátí zparsovanou hodnotu stringu s2 podle pravidla s1 gramatiky x	ParseResult
x.rule(s)	Vrátí pravidlo se jménem s gramatiky x	BNFRule

Tabulka 14 - Metody nad objekty typu BNFRule

Jméno metody	Popis	Výsledek
x.parse()	Vrátí zparsovanou hodnotu textové hodnoty nebo atributu podle pravidla x	ParseResult

Tabulka 15 - Metody nad objekty typu Bytes

Jméno metody	Popis	Výsledek
x = new Bytes(n)	Vrátí pole bytů x o velikosti n. Všechny prvky jsou nastaveny na 0.	Bytes
x.add(n)	Přidá hodnotu n za poslední prvek pole bytů x.	
x.clear()	Vynuluje prvky pole bytů x.	
x.getAt(n)	Vátí n-tý prvek pole x (index prvního prvku je 0).	Int
x.insert(n1, n2)	Vloží byte n2 před prvek n1-tý v poli bytů x (index prvního prvku je 0).	
x.remove(n)	Odstraní z pole bytů x n-tý prvek a vrátí jeho hodnotu (index prvního prvku je 0).	Int
x.setAt(n1, n2)	Nastaví v poli bytů hodnotu n1-tého prvku na hodnotu n2 (index prvního prvku je 0).	
x.toBase64()	Vrátí string s hodnotou pole bytů ve formátu Base64.	String
x.toHex()	Vrátí string s hodnotou pole bytů ve formátu Hexadecimal.	String

Tabulka 16 - Metody nad objekty typu Container

Jméno metody	Popis	Výsledek
x = new Container()	Vytvoří prázdný Container x.	Container
x.addItem(o)	Přidá na konec sekvence prvků Containeru x objekt o.	
x.getElement()	Vrátí první element nalezený v Containeru x (nebo null).	Element
x.getElement(n)	Vrátí n-tý element nalezený v Containeru x (nebo null).	Element
x.getElements()	Vrátí Container s elementy nalezenými v Containeru x.	Container
x.getElements(s)	Vrátí Container s elementy se jménem s nalezenými v Containeru x.	Container
x getItemType(n)	Vrátí type ID n-tého prvku v Containeru x.	Int
x.getLength()	Vrátí počet sekvenčních prvků v Containeru x.	int
x.getNamedItem(s)	Vrátí hodnotu pojmenovaného prvku v Containeru x.	AnyValue
x. getNamedString(s)	Vrátí hodnotu pojmenovaného prvku v Containeru x jako string.	String

x.getText()	Vrátí string složený z prvků typu string v Containeru x.	String
x.getText(n)	Vrátí string s n-tým prvkem typu string v Containeru x.	String
x.hasNamedItem(s)	Vrátí true když Container x má pojmenovaný prvek se jménem s.	Boolean
x.isEmpty()	Vrátí true když Container x nemá žádné prvky.	boolean
x.item(n)	Vrátí n-tý prvek Containeru x.	AnyValue
x.removeItem(n)	Odstraní n-tý prvek z Containeru x.	
x.removeNamedItem (s)	Odstraní pojmenovaný prvek se jménem s z Containeru x.	
x.set NamedItem (v)	Uloží pojmenovanou požku v do Containeru x.	
x.set NamedItem (s, v)	Vytvoří pojmenovanou položku se jménem s a hodnotou v a uloží ji do Containeru x.	
x.sort ()	Vrátí vzestupně setříděný Container x.	Container
x.sort (s)	Vrátí vzestupně setříděný Container x. U elementů aplikuje jako klíč výsledek XPath výrazu s.	Container
x.sort (s, b)	Vrátí setříděný Container x. U elementů aplikuje jako klíč výsledek XPath výrazu ze strigu s. Je-li b true třídění je vzestupné, jinak sestupné	Container
x.toElement ( )	Vytvoří element z Containeru x.	Element
x.toElement (s)	Vytvoří element se jménem s z Containeru x.	Element
x.toElement (s1, s2)	Vytvoří element s namespace URI s1 a jménem s2 z Containeru x.	Element

Tabulka 17 - Metody nad objekty typu Datetime (datum a čas)

Jméno metody	Popis	Výsledek
x = new Datetime(s)	vytvoří objekt x s datumem ze stringu s, který musí být ve tvaru ISO.	Datetime
x.addDay(i)	přičte k datumu d počet dní i (i může být i záporné, pak se dny odečtou) a vrátí novou hodnotu	Datetime
x.addHour(i)	přičte k datumu d počet hodin i (i může být i záporné, pak se hodiny odečtou) a vrátí novou hodnotu	Datetime
x.addMillisecond(i)	přičte k datumu d počet milisekund i (i může být i záporné, pak se milisekundy odečtou) a vrátí novou hodnotu	Datetime
x.addMinute(i)	přičte k datumu d počet minut i (i může být i záporné, pak se minuty odečtou) a vrátí novou hodnotu	Datetime
x.addMonth(i)	přičte k datumu d počet měsíců i (i může být i záporné, pak se měsíce odečtou) a vrátí novou hodnotu	Datetime
x.addNanosecond(n)	přičte k datumu d počet nanosekund n (n může být i záporné, pak se nanosekundy odečtou) a vrátí novou hodnotu	Datetime
x.addSecond(i)	přičte k datumu d počet sekund i (i může být i záporné, pak se sekundy odečtou) a vrátí novou hodnotu	Datetime
x.addYear(i)	přičte k datumu d počet roků i (i může být i záporné, pak se roky odečtou) a vrátí novou hodnotu	Datetime
x.easterMonday(y)	Vrátí datum s velikonočním pondělím roku z datumu a vrátí novou hodnotu	Datetime
x.getDay()	vrátí den z datumu	int
x.getFractionalSecond()	vrátí z datumu hodnotu vteřin včetně zlomkové části	float
x.getHour()	vrátí hodiny z datumu	int
x.getMillisecond()	vrátí počet milisekund od začátku dne	int
x.getMinute()	vrátí minuty z datumu	int
x.getMonth()	vrátí měsíc z datumu d (leden je 1)	int
x.getNanosecond()	vrátí počet nanosekund od začátku dne	int
x.getSecond()	vrátí sekundy z datumu	int
x.getWeekDay()	vrátí den v týdnu z datumu (1 je neděle, 7 je sobota)	int
x.getYear()	vrátí rok z datumu	int

x.getZoneName()	vrátí jméno časové zóny datumu.	String
x.getZoneOffset()	vrátí z datumu posun časové zóny vůči nultému poledníku v milisekundách	int
x.isLeapYear()	vrátí true, je-li rok datumu přestupný.	boolean
x.lastDayOfMonth()	vrátí poslední den v měsíci datumu.	int
x.setDay(i)	nastaví v datumu den i a vrátí novou hodnotu	Datetime
x.setDaytimeMillis(i)	nastaví čas aktuálního dne v datumu podle počtu milisekund v parametru i a vrátí novou hodnotu	Datetime
x.setHour(i)	nastaví v datumu hodinu i a vrátí novou hodnotu	Datetime
x.setMillisecond(i)	nastaví v datumu milisekundu i a vrátí novou hodnotu	Datetime
x.setMinute(i)	nastaví v datumu minutu i a vrátí novou hodnotu	Datetime
x.setMonth(i)	nastaví v datumu měsíc i a vrátí novou hodnotu	Datetime
x.setSecond(i)	nastaví v datumu sekundu i a vrátí novou hodnotu	Datetime
x.setYear(i)	nastaví v datumu rok i a vrátí novou hodnotu	Datetime
x.setZoneName(s)	nastaví v datumu jméno časové zóny na s a vrátí novou hodnotu	Datetime
x.setZoneOffset(i)	nastaví v datumu posun časové zóny (i jsou milisekundy) a vrátí novou hodnotu	Datetime
x.toMillis()	vrátí hodnotu, která odpovídá počtu milisekund od 1. ledna 1970	int
x.toString(s)	vrátí znakový řetězec s datumem podle masky s	String

Tabulka 18 - Metody nad objekty typu Duration (časový interval)

Jméno metody	Popis	Výsledek
new Duration(s)	Konstruktor intervalu Duration. Vytvoří objekt podle stringu s, který musí být ve formátu ISO	Duration
x.getDays()	Vrátí počet dní intervalu	int
x.getEnd	Vrátí datum s koncem intervalu	Datetime
x.getFractionalSecond()	Vrátí počet vteřin včetně zlomkové části	float
x.getHours()	Vrátí počet hodin	int
x.getMinutes()	Vrátí počet minut	int
x.getMonths)	Vrátí počet měsíců	int
x.getNextDate()	Vrátí následující datum a čas	Datetime
x.getRecurrence()	Vrátí počet opakování intervalu	int
x.getSeconds()	Vrátí počet vteřin	int
x.getStart()	Vrátí počáteční datum a čas intervalu	Datetime
x.getYears()	Vrátí počet vteřin včetně zlomkové čísti	float

Tabulka 19 - Metody nad objekty typu XML Element

Jméno metody	Popis	Výsledek
new Element(s)	Konstruktor elementu. Vytvoří element se jménem s	Element
new Element(s1, s2)	Konstruktor elementu. Vytvoří element s jmenným prostorem s1 a jménem s.	Element
x.addElement(e)	Přidá k elementu x na aktuální konec seznamu potomků element e. Pokud je rodič kořen XML stromu, ohlásí se chyba	
x.addText(s)	Přidá k elementu x na aktuální konec seznamu potomků textov7 uzel s hodnotou s. Pokud je rodič kořen XML stromu, ohlásí se chyba	
x.getAttribute(s)	Vrátí hodnotu atributu s v elementu x.	String
x.getAttribute(s1, s2)	Vrátí hodnotu atributu s namespace s1 a lokálním jménem s2 v elementu x.	String
x.getChildNodes()	Vrátí Container se seznamem potomků elementu x.	Container
x.getNamespaceURI()	Vrátí string s namespace URI elementu x.	String

x.getTagName()	Vrátí kvalifikované jméno elementu x.	String
x.getText()	Vrátí texto9vou hodnotu elementu x.	String
x.hasAttribute (s)	vrátí true pokud element x má atribut se jménem s	boolean
x.hasAttributeNS (s1, s2)	vrátí true pokud element x má atribut se jménem s2 a namespace s1.	Boolean
x.isEmpty()	Vrátí true pokud element x nemá potomky.	Boolean
x.setAttribute(s1, s2)	Nastaví v elementu x atribut se jménem s1 a hodnotou s2.	
x.setAttribute(s1, s2, s3)	Nastaví v elementu x atribut s namespace URI s1, jménem s2 a hodnotou s3.	
x.toContainer() *nebo* *x.toContext ()*	vrátí Container vytvořený z elementu x. *Poznámka: Z důvodu kompatibility je možné použít i metodu toContext.*	Container
x.toString(b)	Vrátí string vytvořený z elementu x. Je-li b true, pak string je indentovaný tvar elementu.	String

Tabulka 20 - Metody nad objekty typu Exception

Jméno metody	Popis	Výsledek
x=new Exception(s)	Vytvoří Exception s textem s	Exception
x=new Exception(s1,s2)	Vytvoří Exception s reportem s ID s1 a textem s2.	Exception
x=new Exception(s1,s2,s3)	Vytvoří Exception s reportem s ID s1, textem s2 parametry s3.	Exception
x.getReport()	Vrátí report z Exception x	Report
x.getMessage()	Vrátí text z Exception x	String

Tabulka 21 - Metody nad objekty typu Input (vstupní stream)

Jméno metody	Popis	Výsledek
x=new Input(s)	Vytvoří input stream podle jména s.	Input
x=new Input(s, b)	Vytvoří input stream podle jména s. Je-li b true, bude se číst ve formátu XML	Input
x=new Input(s1, s2)	Vytvoří input stream podle jména s1 a s kódováním s2.	Input
x=new Input(s1, s2, b)	Vytvoří input stream podle jména s1 a s kódováním s2. Je-li b true, bude se číst ve formátu XML	Input
x.eof()	Vrátí true, pokud je Input x na konci	boolean
x.readln()	Přečte řádek z Input.	String

Tabulka 22 - Metody nad objekty typu NamedValue (pojmenovaná hodnota)

Jméno metody	Popis	Výsledek
x=new NamedValue(s, v)	Vytvoří pojmenovanou hodnotu x se jménem s a hodnotou v.	NamedValue
x.getName()	Vrátí jméno pojmenované položkz x.	String
x.getValue()	Vrátí hodnotu pojmenované položky x.	AnyValue
x.setName (s)	Nastaví jméno s pojmenované položky x.	

Tabulka 23 - Metody nad objekty typu Output (výstupní stream)

Jméno metody	Popis	Výsledek
x=new Output(s)	Vytvoří output stream podle jména s.	Output
x=new Output(s, b)	Vytvoří output stream podle jména s. Je-li b true, bude se zapisovat ve formátu XML.	Output
x=new Output(s1, s2)	Vytvoří output stream podle jména s1 a kódovou stránku s2.	Output
x=new Output(s1, s2, b)	Vytvoří output stream podle jména s1 a kódovou stránku s2. Je-li b true, bude se zapisovat ve formátu XML.	Output
x.error(s)	Zapíše se chybový záznam s textem s.	
x.error(s1, s2)	Zapíše se chybový záznam s kódem s1 a textem s2.	

x.error(s1, s2, s3)	Zapíše se chybový záznam s textem s1, textem s2 a parametry s3.	
x.getLastError()	Vrátí posledně zapsaný záznam error.	Report
x.out(s)	Zapíše text s do x.	
x.outln()	Zapíše novou řádku do x.	
x.outln(s)	Zapíše do x text s a novou řádku.	
x.printf(s, v1, ...)	Zapíše do x a string vytvořený z hodnot parametrů v1, ... podle masky s (viz metoda printf v java.io.PrintStream).	
x.printf(l, s, v1, ...)	Zapíše do x string vytvořený z hodnot parametrů v1, ... podle oblasti v parametru l (typu Locale) a podle masky s (viz metoda printf v java.io.PrintStream).	
x.putReport(r)	Zapíše do x Report r.	

Tabulka 24 - Metody nad objekty typu ParseResult

Jméno metody	Popis	Výsledek
x=new ParseResult(s)	Vytvoří parsovanou hodotu x ze stringu s.	ParseResult
x.booleanValue()	Vrátí parsovanou boolean hodnotu z x.	boolean
x.bytesValue()	Vrátí parsované byty z x.	Bytes
x.matches()	Vrátí true pokud parsovaná hodnota x neobsahuje chyby. Jinak vrátí false	boolean
x.datetimeValue()	Vrátí parsované datum z x.	Datetime
x.durationValue()	Vrátí parsovaný interval z x.	Duration
x.decimalValue()	Vrátí parsovanou hodnotu Decimal z x.	Decimal
x.error (s)	Nastaví chybové hlášení s textem s do x	
x.error (s1, s2)	Nastaví chybové hlášení s identifikátorem s1 a textem s2 do x	
x.error (s1, s2, s3)	. Nastaví chybové hlášení s identifikátorem s1, textem s2 a parametry s3do x	
x.floatValue()	Vrátí parsovanou hodnotu float z x.	float
x.getError()	Vrátí chybové hlášení z x.	Report
x.getParsedString()	Vrátí parsovaný text z x.	String
x.getValue()	Vrátí parsovanou hodnotu z x.	AnyValue
x.setParsedString (s)	Nastaví hodnotu parsovanéhoí strigu s do x.	
x.set Value(v)	Nastaví parsovanou hodnotu v do x.	

Tabulka 25 - Metody nad objekty typu Regex

Jméno metody	Popis	Výsledek
new Regex(s)	Vytvoří regulární výraz podle stringu s	Regex
x.getMatcher(s)	Vrátí RegexResult podle regulárního výrazu x ze stringu s	RegexResult
x.matches(s)	Vrátí true z jestli byl splněn regulární výraz x na stringu s	boolean

Tabulka 26 - Metody nad objekty typu RegexResult

Jméno metody	Popis	Výsledek
x.end(n)	Vrátí koncový index skupiny n	int
x.group(n)	Vrátí string ze skupiny n	String
x.groupCount ()	Vrátí počet skupin	int
x.matches ()	Vrátí true, jestli byl splněn regulární výraz	boolean
x.start (n)	Vrátí počáteční index skupiny n	int

Tabulka 27 - Metody nad objekty typu Report

Jméno metody	Popis	Výsledek
x=new Report(s)	Vytvoří report s textem s.	Report

x=new Report(s1,s2)	Vytvoří report s ID s1 a textem s2.	Report
x=new Report(s1,s2,s3)	Vytvoří report s ID s1, textem s2 a modifikací s3.	Report
x.getParameter(s)	Vrátí string s parametrem s z reportu x.	String
x.setParameter(s1, s2)	Vrátí Report, kde nastaví v reportu x v modifikačním stringu parametr se jménem s1 na hodnotu s2.	Report
x.setType(s)	Vrátí Report, kde nastaví typ reportu podle parametru s: "E" … error "W" … warning "F" … fatal error "I" … informační zápis "M"... zápis message "T" … text	Report

Tabulka 28 - Metody nad objekty typu ResultSet

Jméno metody	Popis	Výsledek
x.close()	Uzavře ResultSet x.	
x.closeStatement()	Uzavře statement spojený s ResultSet x.	
x.getCount()	Vrátí počet prvků v ResultSet x.	Int
x.getItem()	Vrátí aktuální prvek z ResultSet x.	String
x.getItem(s)	Vrátí prvek pojmenovaný s z ResultSet x.	String
x.hasItem(s)	Vrátí true, když pojmenovaný s v ResultSet x existuje.	boolean
x.hasNext()	Vrátí true, když existuje další řádek v ResultSet x.	boolean
x.isClosed()	Vrátí true, když ResultSet x je uzavřený.	boolean
x.next()	Nastaví další řádek v ResultSet x a vrátí true, když existuje.	boolean

Tabulka 29 - Metody nad objekty typu Service

Jméno metody	Popis	Výsledek
x=new Service(s1,s2,s3,s4)	Vytvoží objekt x umožňující přístup ke službám databáze. Parametr s1 je druh databáze (např "jdbc"), s2 je URL, s3 je user a s4 je heslo.	Service
x.close()	Uzavre databázi x.	
x.commit()	Provede operaci commit v databázi přístupné přes x	
x.execute(s1, …)	Provede príkaz s1 s parametry s2, …v databázi přístupné přes x. Vrátí true, když se příkaz provede.	boolean
x.hasItem(s1,…)	Vrátí true, když prvek definovaný parametry existuje.	boolean
x. isClosed()	Vrátí true, když databáze přístupná přes x je uzavřena.	boolean
x. prepareStatement(s)	Připraví a vrátí příkaz s v databázi přístupná přes x.	Statement
x.query(s1, s2)	Provede dotaz do databáze přístupná přes x a vrátí ResultSet objekt.	ResultSet
x.queryItem(s1, s2, s3)	Provede dotaz do databáze přístupná přes x a vrátí string s prvkem s3t.	String
x.rollback()	Provede operaci rollback v databázi přístupná přes x.	
x.setProperty(s1, s2)	Nastaví propery s1 na hodnotu s2 v databázi přístupná přes x. Vrátí true, pokud nastavení proběhlo.	boolean

Tabulka 30 - Metody nad objekty typu Statement

Jméno metody	Popis	Výsledek
x.close()	Uzavře příkaz x.	
x.execute(s1, …)	Provede příkaz s1, … a vrátí true, pokud byl proveden.	boolean
x.hasItem(s1, …)	Vrátí true, když existuje prvek podle parametrů s1, …	boolean
x.isClosed()	Vrátí true, když príkaz x byl uzavřen.	boolean
x.query(s1, …)	Provede dotaz s parametry s1, … a vrátí ResultSet s výsledkem.	ResultSet

x.queryItem(s1, s2…)	Provede dotaz s na prvek s1 parametry s2, … a vrátí ResultSet s výsledkem.	ResultSet

Tabulka 31 - Metody nad objekty typu String

Jméno metody	Popis	Výsledek
x.contains(s)	Vrátí true, když string x obsahuje znakový řetězec s.	boolean
x.containsi(s)	Vrátí true, když string x obsahuje znakový řetězec s bez ohledu na velká/malá písmena.	boolean
x.cut(n)	Zkrátí string x na maximální délku n.	String
x.endsWith(s)	Vrátí true, když string x končí znakovým řetězcem s.	boolean
x.endsWithi(s)	Vrátí true, když string x končí znakovým řetězcem s bez ohledu na velká/malá písmena.	boolean
x.equals(s)	Vrátí true, když string x má stejnou hodnotu jako s bez ohledu na velká/malá písmena.	boolean
x.equalsIgnoreCase(s)	Vrátí true, když string x má stejnou hodnotu jako s.	boolean
x.getBytes()	Vrátí pole bytů vytvořené ze stringu s (v aktuálním kódování systému)	Bytes
x.getBytes(s)	Vrátí pole bytů vytvořené ze stringu s (v kódování systému podle kódové stránky s názvem s)	Bytes
x.indexOf(s)	Vrátí pozici výskytu znakového řetězce s ve stringu x. Pozice začíná od 0 a pokud řetězec neexistuje, vrátí se -1.	int
x.indexOf(s, n)	Vrátí pozici výskytu znakového řetězce s ve stringu x počínaje pozicí n. Pozice začíná od 0 a pokud řetězec neexistuje, vrátí se -1	int
x.isEmpty()	Vrátí true, pokud string s je prázdný.	boolean
x.lastIndexOf(s)	Vrátí pozici posledního výskytu znakového řetězce s ve stringu x. Pokud řetězec není nalezen, vrátí se -1.	int
x.lastIndexOf(s,n)	Vrátí pozici posledního výskytu znakového řetězce s ve stringu x před pozicí n. Pokud řetězec není nalezen, vrátí se -1.	int
x.length()	Vrátí počet znaků ve stringu x.	int
x.startsWith(s)	Vrátí true, když string x začíná znakovým řetězcem s.	boolean
x.startsWithi(s)	Vrátí true, když string x začíná znakovým řetězcem s bez ohledu na velká/malá písmena.	boolean
x.substring(n)	Vrátí část stringu x počínaje pozicí n do konce.	String
x.substring(n1, n2)	Vrátí část stringu x počínaje od začátku do pozice n.	String
x.toLower()	Vrátí string, ve kterém jsou všechna velká písmena ve stringu x z nahrazena malými písmeny.	String
x.toUpper()	Vrátí string, ve kterém jsou všechna malá písmena ve stringu x z nahrazena velkými písmeny.	String
x.trim()	Vrátí string, ve kterém jsou odstraněny mezery, tabelátory a nové řádky na začátku a na konci stringu x.	String

Tabulka 32 - Metody nad objekty typuXmlOutStream

Jméno metody	Popis	Výsledek
x=new XmlOutStream(s)	Vytvoří objekt XmlOutStream se jménem s.	XmlOutStream
x.setIndenting (b)	Je-li b true, zápis se provádí indentovaně.	
x.writeElementStart (e)	Zapíše začátek element e.	
x.writeElementEnd ()	Zapíše konec elementu.	
x.writeElement (e)	Zapíše element e.	
x.writeText (s)	Zapíše text s.	
x. close ()	Ukončí zápis.	

3.2.22 Matematické metody

Ve skriptu je možné použít matematické metody z knihovny "java.lang.Math". Tyto metody jsou implementovány jak pro typ skriptu "float", tak i "int", který se v případě potřeby převede na "float". Výsledek je buď "float", nebo "int" podle typu metody. Uvědomme si, že typ "int" ve skriptu je vždy implementován jako "long" a typ "float" je implementován jako "double".

Tabulka 33 - Metody matematických funkcí převzatých z java.lang.Math

Jméno metody	Popis	Výsledek
abs(x)	Viz metodu java.lang.Math.abs	int nebo float
acos(x)	Viz metodu java.lang.Math.acos	float
asin(x)	Viz metodu java.lang.Math.asin	Float
atan(x)	Viz metodu java.lang.Math.atan	float
atan2(x, y)	Viz metodu java.lang.Math.atan2	float
cbrt (x)	Viz metodu java.lang.Math.cbrt	float
ceil(x)	Viz metodu java.lang.Math.ceil	float
cos(x)	Viz metodu java.lang.Math.cos	float
cosh(x)	Viz metodu java.lang.Math.cosh	float
exp(x)	Viz metodu java.lang.Math.exp	float
expm1(x)	Viz metodu java.lang.Math.expm1	float
floor(x)	Viz metodu java.lang.Math.floor	float
hypot(x, y)	Viz metodu java.lang.Math.hypot	float
IEEEremainder(x, y)	Viz metodu java.lang.Math. IEEEremainder	float
log(x)	Viz metodu java.lang.Math.log	float
log10(x)	Viz metodu java.lang.Math.log10	float
log1p(x)	Viz metodu java.lang.Math.log1p	float
max(x,y)	Viz metodu java.lang.Math.max	int nebo float
min(x,y)	Viz metodu java.lang.Math.min	float
pow(x, y)	Viz metodu java.lang.Math.pow	float
rint(x)	Viz metodu java.lang.Math.rint	float
round(x)	Viz metodu java.lang.Math.round	int
signum(x)	Viz metodu java.lang.Math.signum	float
sin(x)	Viz metodu java.lang.Math.sin	float
sinh(x)	Viz metodu java.lang.Math.sinh	float
sqrt(x)	Viz metodu java.lang.Math.sqrt	float
tan(x)	Viz metodu java.lang.Math.tan	float
tanh(x)	Viz metodu java.lang.Math.tanh	float
toDegrees(x)	Viz metodu java.lang.Math.toDegrees	float
toRadians(x)	Viz metodu java.lang.Math.toRadians	float
ulp (x)	Viz metodu java.lang.Math.ulp	float

Pro práci s typem "Decimal" (je implementován jako java.math.BigDecimal) jsou implementovány metody tohoto typu:

Tabulka 34 - Metody matematických funkcí převzatých z java.math.BigDecimal

Jméno metody	Popis	Výsledek
decimalValue(x)	Konstruktor; x může být int, double, String nebo BigDecimal.	Decimal
abs(x)	Viz java.math.BigDecimal.abs	Decimal
add(x,y)	Viz java.math.BigDecimal.add	Decimal
compare(x,y)	Viz java.math.BigDecimal.compare	int

divide(x,y)	Viz java.math.BigDecimal.divide	Decimal
equals(x,y)	Viz java.math.BigDecimal.equals	boolean
intValue(x)	Viz java.math.BigDecimal.longValue	int
floatValue(x)	Viz java.math.BigDecimal.doubleValue	float
max(x,y)	Viz java.math.BigDecimal.max	Decimal
min (x,y)	Viz java.math.BigDecimal.min	Decimal
movePointLeft(x, n)	Viz java.math.BigDecimal.movePointLeft	Decimal
movePointRight(x, n)	Viz java.math.BigDecimal.movePointRight	Decimal
multiply (x, y)	Viz java.math.BigDecimal.multiply	Decimal
negate(x)	Viz java.math.BigDecimal.negate	Decimal
plus(x)	Viz java.math.BigDecimal.plus	Decimal
pow(x, n)	Viz java.math.BigDecimal.pow	Decimal
remainder(x)	Viz java.math.BigDecimal.remainder	Decimal
round(x)	Viz java.math.BigDecimal.round	Decimal
scaleByPowerOfTen(x,n)	Viz java.math.BigDecimal.scaleByPowerOfTen	Decimal
setScale (x,n)	Viz java.math.BigDecimal. setScale	Decimal
stripTrailingZeros(x)	Viz java.math.BigDecimal.stripTrailingZeros	Decimal
subtract(x,y)	Viz java.math.BigDecimal.subtract	Decimal
ulp(x)	Viz java.math.BigDecimal. ulp	Decimal

3.2.23 Externí metody

Kromě implementovaných metod a uživatelských metod je možné ve skriptu volat i externí metody, které uživatel napíše sám v jazyce Java. Externí metody musí být deklarovány jako statické. Tyto metody musí splňovat určité konvence. Jsou dva způsoby a formáty předávání parametrů. Existují tři způsoby předávání parametrů externím metodám:

a) externí procedury s parametry, které korespondují se seznamem parametrů volající metody ve skriptu. Deklarace metody s pevně stanovenými parametry má tvar jako běžná statická metoda. Parametry metody odpovídají konkrétnímu volání metody skriptu. Typy hodnot jednotlivých prvků pole jsou určeny níže uvedenou tabulkou:

Tabulka 35 - Typy hodnot předávaných externím uživatelským metodám

Jméno typu ve skriptu	Typ předaný externí metodě
Boolean	Logická hodnota ("boolean")
Datetime	datum a čas ("java.util.Calendar")
float	číslo v pohyblivé řádové čárce ("double")
int	celé číslo ("long")
Regex	předkompilovaný regulární výraz (podle verze Java prostředí: pro verzi JDK 1.3 je použita knihovna Apache a pak je objekt typu "org.apache.oro.text.regex.Pattern". Od verze JDK 1.4 a výše je regulární výraz součástí standardní knihovny a regulární výraz je v tomto případě objekt typu "java.util.regex.Pattern")
RegexResult	výsledek regulárního výrazu. Jako u předchozího případu se typ objektu liší podle verze Javy. Pro verzi JDK 1.3 je použita knihovna Apache a objekt je typu "org.apache.oro.text.regex.MatchResult" a od verze 1.4 výše je použit objekt "java.util.regex.Matcher")
String	znakový řetězec ("java.lang.String")
Element	element ("org.w3c.dom.Element")
Bytes	Pole bytů ("byte[]")
Context	Pole XDValue. Prvky může být sekvence libovoného typu skriptu. Kromě toho je možné do tohoto typu uložit pojmenované hodnoty. Na prvky sekvence se obracíme indexem, na pojmenované hodnoty jménem.

Příklad externích metod v jazyce Java:

```
public static boolean tab(String tabName, String colName, String value) {
...
}

public static void chyba(int kod) {
...
}
```

Ve skriptu mohou být tyto metody volány např.:

```
<elem value = "required tab('tabulka','sloupec', getText()) onFalse chyba(123);" />
```

b) externí procedury s polem parametrů. Tyto procedury mají jediný parametr s polem prvků typu "cz.syntea.xdef.XDValue[]". Počet prvků v tomto poli závisí na konkrétním volání metody skriptu. Metoda tohoto typu je překladačem volána tehdy, když je nalezena, ale když není nalezena příslušná metoda s odpovídajícím pevným seznamem parametrů. Parametry jsou uloženy do pole objektů, jejichž počet a typ odpovídá volanému seznamu. Příklad deklarace externí metody:

```
public static boolean MyMethod(XDValue[] params) {
  int pocetParametru = params.length;
  for (int i = 0; i < params.length; i++) {
    switch(params[i].getItemType) {
      case XDValueType.BOLEAN:
         break;
      case XDValueType.INT:
        ...
  }
...
}
```

c) Externí metody s prvním parametrem typu XXNode, XXElement, XXData. Tyto metody umožňují použití odpovídajících metod nad tímto parametrem. Používají se zejména k parsování hodnot textu v textových uzlech nebo atributech (text získají metodou getText). Ve skriptu se první parametr neuvádí, dosadí se automaticky. Ukázka:

```
public static XDParseResult MujTyp(XDData data) {
  String s = data.getText();
  XDParseResult result = XDFactory.createParseResult(s);
  if (chyba) {
    result.error("….");
  }
  return result;
}
```

Ve skriptu se metoda volá bez prvního parametru, který se dosadí automaticky: `x="required MujTyp}()"`

Externí metody lze deklarovat v elementu <xd:declaration> příkazem "external method", za nímž následuje popis externí metody, který se skládá z následujících částí:

- **Typ výsledku** je nekvalifikované jméno v Java tvaru - např. "String", "long", "XDService" atd.
- **Kvalifikované jméno metody** - např. "cz.projekt.Trida.metoda".
- V závorkách je uvedený **seznam typů parametrů** oddělených čárkami - např.

  "(cz.syntea.xd.XXElement, String, long)". Seznam může být prázdný a jména typů jsou nekvalifikovaná.
- může následovat nepovinně klíčové jméno "as" následované alias jménem metody (přes toto jméno se metoda bude volat v X-definici). Pokud "as" a jméno není uvedeno, použije se jméno metody, jak je deklarováno v Javě.
- popis je ukončen středníkem.

Příklad:

```
<xd:declaration>
  external method boolean cz.projekt.Trida.MyMethod(XXData);
  external method XDParseResult cz.projekt.Trida.MujTyp(XDData);
  ...
</xd:declaration>
```

Pokud je metod více, je možné jejích popis zapsat do bloku {mezi složené zárorky} Např:

```
<xd:declaration>
```

```
  external method {
    boolean cz.projekt.Trida.MyMethod(XXValue[]);
    XDParseResult cz.projekt.Trida.MujTyp(XDData);
    ...
  }
</xd:declaration>
```

3.2.24 Volby (options)

Část skriptu, ve které jsou volby, je nepovinná. Pokud je vynechána, převezmou se volby, které jsou přednastaveny v X-definici, která byla použita při zpracování kořenového XML elementu. Pokud je uvedena v rámci definice elementu, atributu, nebo textového uzlu, nastaví se příslušné hodnoty podle uvedené specifikace.

Zápis voleb je uveden klíčovým slovem "options", za nímž následuje seznam jmen voleb, oddělených čárkou. Jména voleb jsou v následující tabulce.

Tabulka 36 - Volby (options)

Jméno volby	Popis
acceptEmptyAttributes	Prázdné atributy se zkopírují ze vstupu (bez ohledu na jejich typ a na to, jestli jsou deklarované jako *optional*).
preserveAttrWhiteSpaces	Nadbytečné mezery v atributech jsou ponechány. Defaultní ?
preserveComments	Komentáře jsou kopírovány do výsledného dokumentu (ve validačním režimu). Volba je přípustná pouze v záhlaví X-definice.
preserveEmptyAttributes	Prázdné atributy jsou ponechány, jen pokud je příslušný atribut nastaven jako *optional*.
preserveTextWhiteSpaces	Nadbytečné mezery v textových uzlech elementů jsou ponechány.
ignoreAttrWhiteSpaces	Nevýznamné nadbytečné mezery v atributech jsou před dalším zpracováním odstraněny.
ignoreComments	Komentáře jsou ignorovány. Volba je přípustná pouze v záhlaví X-definice. Defaultní nastavení.
ignoreEmptyAttributes	Prázdné atributy (kde délka hodnoty je nulová) jsou ignorovány (předtím jsou provedeny operace, týkající se odstranění mezer). Defaultní nastavení.
ignoreTextWhiteSpaces	Nadbytečné mezery v textových uzlech jsou před dalším zpracováním odstraněny.
preserveAttrCase	Velikost písmen v atributech zůstává nezměněna. Defaultní nastavení.
preserveTextCase	Velikost písmen hodnot textových uzlů elementu zůstává nezměněna. Defaultní nastavení.
setAttrLowerCase	Písmena v atributech jsou před dalším zpracováním nastavena na malá.
setAttrUpperCase	Písmena v atributech jsou před dalším zpracováním nastavena na velká.
setTextLowerCase	Písmena hodnot textových uzlů elementu jsou před dalším zpracováním nastavena na malá.
setTextUpperCase	Písmena v textových uzlech jsou před dalším zpracováním nastavena na velká.
trimAttr	Mezery, tabelátory a nové řádky se před dalším zpracováním odstraní na začátku a na konci hodnot atributů. Defaultní hodnota.
noTrimAttr	Mezery, tabelátory a nové řádky na začátku a na konci hodnot atributů jsou ponechány.
trimText	Mezery, tabelátory a nové řádky se před dalším zpracováním odstraní na začátku a na konci hodnot textových uzlů elementu. Deaultní nastavení.
noTrimText	Mezery, tabelátory a nové řádky na začátku a na konci hodnot textových uzlů elementu jsou ponechány.
moreAttributes	V elementu je povolen výskyt i nedeklarovaných atributů. Tyto atributy jsou zkopírovány do výsledného dokumentu.
moreElements	V elementu je povolen výskyt i nedeklarovaných elementů. Tyto elementy jsou zkopírovány do výsledného dokumentu.
moreText	V elementu je povolen výskyt i nedeklarovaných textových uzlů. Tyto uzly jsou zkopírovány do výsledného dokumentu.
clearAdoptedForgets	Pokud je tato volba specifikována ve skriptu elementu, jsou všechny akce "forget"

	zrušeny pro všechny vnořené elementy a jejich potomky.
ignoreEntities	Option je ve skriptu X-definice a její specifikace způsobí ignorování externích souborů s entitami (v DTD souborech). Volba se převezme z X-definice, která byla použita jako zahájení zpracování.
resolveEntities	Option je ve skriptu X-definice a její specifikace způsobí standardní zpracování externích souborů s entitami (v DTD souborech). Volba se převezme z X-definice, která byla použita jako zahájení zpracování. Defaultní nastavení.
resolveIncludes	Option je ve skriptu X-definice a její specifikace způsobí, že jsou řešeny odkazy na vnější soubory pomocí elementů include s namespace http://www.w3.org/2001/XInclude. Defaultní nastavení.
ignoreIncludes	Option je ve skriptu X-definice a její specifikace způsobí, že nejsou řešeny odkazy na vnější soubory pomocí elementů include s namespace "http://www.w3.org/2001/XInclude".
acceptQualifiedAttr	u atributu, ktery je deklarovan bez namespace může byt atribut zapsán i s prefixem jmenného prostoru elementu, v němž je tento atribut deklarován. Defaultní nastavení.
notAcceptQualifiedAttr	Kvalifikovany atribut není povolen
Nillable	Element může být prázdný, pokud má kvalifikovaný atribut nill" s hodnotou true". Jmenný prostor atributu musí být: "http://www.w3.org/2001/XMLSchema-instance". Volba umožňuje kompatibititu s vlastností "nillable" v XML schematech.
noNillable	Element není nillable. Defaultní nastavení.
acceptOther	Při validaci XML jsou výskyty objektů nedeklarovaných v příslušném modelu elementu vloženy do výsledného XML dokumentu. Defaultně tato volba není nastavena.
ignoreOther	Při validaci XML jsou výskyty objektů nedeklarovaných v příslušném modelu elementu ignorovány. Defaultně tato volba není nastavena.
cdata	Volba způsobí, že textový uzel je generován jako CDATA sekce. Tato volba je přípustná jen ve skriptu textových uzlů. Defaultně tato volba není nastavena.

Příklad:

```
xd:script = "options ignoreAttrWhiteSpaces, ignoreTextWhiteSpaces, preserveEmptyAttributes"
```

Pozn.: Option "noTrimText" je nutné používat obezřetně. Např. pro následující X-definici:

```
<xd:def xmlns:xd = "http://www.syntea.cz/xdef/3.1" xd:name = "a" xd:root = "E">
 <E xd:script="options noTrimText;">
        <O xd:script="*"/>
        optional string();
 </E>
</xd:def>
```

bude následující následující vstupní soubor validován nekorektně:

```
<E>
        <O/>
</E>
```

Důvodem je skutečnost, že za počátečním element <E> se nachází prázdný řádek (který se před procesem validace neodstraní kvůli nastavenému přepínači "noTrimText"). Prázdný řádek je proto chápán jako hodnota textového uzlu, a protože element <O> má povoleno nula výskytů, předpokládá aparát X-definic, že v elementu <E> se element <O> nevyskytuje (příslušný model v X-definici proto přeskočí a následuje model pro textový uzel) a provede validaci textového uzlu. Aparát následně ze vstupního souboru načte element <O>, který ale při sekvenčním řazení modelů v X-definici pochopitelně nevidí a ohlásí chybu, že se na vstupu neočekávaně objevil element <O> (neočekávaně z toho důvodu, že byl očekáván ještě před zpracováním textového uzlu).

Aparát X-definic tedy chápe výše uvedený vstupní soubor při zapnuté volbě "noTrimText" následovně:

```
<E>
        textový_uzel
        <O/>
        textový_uzel
</E>
```

3.2.25 Odkazy na objekty X-definic

Ve skriptu modelu elementu je možné popsat různé objekty a případně skupiny pomocí odkazů. Odkazy zjednoduší popis podobných objektů a mohou zvýšit přehlednost zápisu i usnadnit údržbu X-definic. Skript umožňuje provést odkaz na model elementu.

3.2.25.1 Odkaz na model elementu

Pokud odkazem popisujeme vnitřní element nějaké struktury, můžeme doplnit pro tento element skript a navíc můžeme přidat i popis dalších atributů. Nespecifikované atributy a části skriptu se převezmou z odkazovaného objektu. V případě, že je cíl odkazu ve stejné X-definici jako odkaz sám, ale jméno se liší, je možné část odkazu se jménem X-definice vynechat:

```
odkaz ::= "ref" jméno_modelu_elementu
```

Příklad použití odkazu:

```
<Osoba jmeno = "required" >
  <Adresa xd:script = "ref Adresa" />
</Osoba>

<Firma nazev = "required " >
  <Adresa xd:script = "ref Adresa" />
</Firma >

<Adresa ulice = "required "
        cislo = "required int()"
        mesto = "required "
        PSC = "required num(5)"
/>
```

V praxi často pracujeme s více X-definicemi. V takovém případě před jméno referovaného modelu elementu musíme napsat jméno X-definice, které oddělíme od jména modelu elementu znakem "#". Tento druh odkazu nazýváme "**rozšířeným odkazem**" a jeho zápis má tvar:

```
ref jmeno_X-definice#jméno_modelu_elementu
```

V následujícím příkladě je element "Pobyt" v X-definici "Osoba" ukázkou rozšířeného odkazu, ve které je navíc odkazovaný model elementu "Adresa" v X-definici "Firma" doplněn o popis dalšího povinného atributu "druh":

```
<xd:def xmlns:xd = "http://www.syntea.cz/xdef/3.1"
        xd:name  = "Osoba"
        xd:root  = "Osoba" >
  <Osoba>
    <Pobyt xd:script = "ref Firma#Adresa"
           druh = "required list('trvala','docasna')"/>
  </Osoba>
</xd:def>

<xd:def xmlns:xd = "http://www.syntea.cz/xdef/3.1"
        xd:name  = "Firma"
        xd:root  = "Firma" >

  <Firma xd:script = "ref Adresa"
         prijmeni    = "illegal"/>

  <Adresa jmeno    = "required string(1,30)"
          prijmeni = "required string(1,30)"
          ulice    = "required string(1,30)"
          cp       = "optional int()"
          mesto    = "required string(1,30)"
          PSC      = "optional num(5)"
          stat     = "required string(2)"
  />
</xd:def>
```

Všimněme si, že v uvedeném příkladu je v elementu <Osoba> přidán navíc oproti odkazovanému modelu "Adresa" atribut "druh". Vidíme, že seznam atributů je možné modifikovat. Pokud chceme naopak některý atribut z odkazovaného modelu zakázat, je možné jej definovat jako "illegal" - viz element <Firma>.

Pokud je v elementu s odkazem uvedena nějaká akce nebo volby, je tato akce nebo volba nahrazena údajem z tohoto elementu.

Poznámka: Dvě X-definice (nebo obdobně více X-definic) můžeme uvést:

a) V rámci jednoho xdef-souboru. V daném případě použijeme element <xd:collection>, který se stane kořenovým elementem a <xd:def> budou jeho podelementy.

```
<xd:collection xmlns:xd = "http://www.syntea.cz/xdef/3.1">

<xd:def xd:name = "Osoba" xd:root = "Osoba" >
          ... /* Obsah X-definice Osoba */
</xd:def>

<xd:def xd:name  = "Firma" xd:root  = "Firma" >
          ... /* Obsah X-definice Firma. */
</xd:def>

<xd:collection>
```

b) Každou X-definici do samostatného xdef-souboru. Poté upravíme volání metody compileXD, které předáme cesty ke všem souborům s X-definicemi, které v rámci daného projektu používáme.

```
XDPool xpool = XDFactory.compileXD(null, "osoba.xdef", "firma.xdef");
```

Pokud jsou v elementu s odkazem deklarovány vnitřní elementy nebo textové hodnoty, je k seznamu potomků z elementu, na nějž odkazuje reference přidán seznam uvedený v elementu s odkazem:

```
<xd:def xmlns:xd = "http://www.syntea.cz/xdef/3.1"
        xd:name  = "Subjekt"
        xd:root  = "Firma | Osoba | Adresa" >

  <Firma xd:script = "ref Adresa ">
        <Nazev> required string(1,30) </Nazev>
  </Firma>

  <Osoba xd:script = "ref Adresa">
    <Jmeno> required string(1,30) </Jmeno>
    <Primeni> required string(1,30) </Primeni>
  </Osoba>

  <Adresa ulice = "required string(1,30)"
          cp = "optional int()"
          mesto = "required string(1,30)"
          PSC = "optional num(5)"
          stat = "required string(2)"
  />
</xd:def>
```

Všimněme si, že v elementu <Firma> je přidán element <Nazev>, a seznam atributů je z elementu <Adresa> převzat.

3.2.25.2 Odkaz na sekvenci potomků modelu elementu

Kromě odkazů na modely elementů je možné se odkazovat i na sekvenci tvořenou potomky elementu. Uveďme příklad:

```
<Manzelstvi Uzavreno = "required date">
   <xd:includeChildNodes ref = "manzele" />
</Manzelstvi>

<manzele>
  <xd:mixed>
   <Manzel   xd:script = "ref Osoba#Osoba" />
   <Manzelka xd:script = "ref Osoba#Osoba" />
  </xd:mixed>
</manzele>
```

Popis potomku elementu <Manzelstvi> se převezme z modelu elementu <manzele>.

Výsledek odpovídá zápisu:

```
<Manzelstvi Uzavreno = "required date">
  <xd:mixed>
   <Manzel   xd:script = "ref Osoba#Osoba" />
   <Manzelka xd:script = "ref Osoba#Osoba" />
  </xd:mixed>
</Manzelstvi>
```

Sekvence elementů, na kterou se odkazujeme, musí být přímým potomkem X-definice, tj. v uvedeném příkladě musí být element <Manzelstvi> přímým potomkem elementu <xd:def>, jinak aparát X-definic příslušnou sekvence nenalezne a ohlásí chybu.

3.2.26 Požadavek porovnání struktury

Pokud chceme porovnat strukturu nějakého elementu s určitým modelem, zapíšeme do skriptu elementu klíčové slovo "implements" nebo "uses" a doplníme odkaz na model, s nímž má být struktura porovnána. Porovnává se pouze část výskytů a typů hodnot jednotlivých prvků, které musí být shodné s modelem v odkazu, zápis dalších sekcí se nesrovnává.

V případě "implements" se musí shodovat i jméno porovnávaného modelu:

```
<xd:collection xmlns:xd = "http://www.syntea.cz/xdef/3.1" >

<xd:def xd:name = "A" >
  <Firma xd:script = "implements B#Firma"
     ulice = "required string(1,30)"
     cp = "optional int()"
     mesto = "required string(1,30)"
     PSC = "optional num(5)"
     stat = "required string(2)">
        <Nazev> required string(1,30) </Nazev>
  </Firma>
</xd:def>

<xd:def xd:name = "B" >
  <Firma xd:script = "ref Adresa ">
        <Nazev> required string(1,30) </Nazev>
  </Firma>
  <Adresa ulice = "required string(1,30)"
          cp = "optional int()"
          mesto = "required string(1,30)"
          PSC = "optional num(5)"
          stat = "required string(2)"
  />
</xd:def>

</xd:collection>
```

V případě příkazu "uses" se jméno modelu s příkazem a v porovnávaném modelu může lisit. Pokud navíc v modelu s odkazem " uses" není deklarována metoda kontroly typu, pak se tato metoda zkopíruje z odkazovaného modelu:

```
<xd:collection xmlns:xd = "http://www.syntea.cz/xdef/3.1" >

<xd:def xd:name = "A" >
  <Firma xd:script = "uses B#Object"
     ulice = "required"
     cp = "optional"
     mesto = "required"
     PSC = "optional"
     stat = "required">
        <Nazev> required </Nazev>
  </Firma>
</xd:def>

<xd:def xd:name = "B" >
  <Object xd:script = "ref Adresa ">
        <Nazev> required string(1,30) </Nazev>
  </Object >
  <Adresa ulice = "required string(1,30)"
          cp = "optional int()"
          mesto = "required string(1,30)"
          PSC = "optional num(5)"
          stat = "required string(2)"
  />
</xd:def>

</xd:collection>
```

3.2.27 Makra ("xd:macro")

Makra slouží ke zjednodušení a zpřehlednění zápisu skriptu a ke snadnější celkové údržbě zdrojového tvaru skriptu v X-definicích. Makro je pojmenovaný objekt, jehož hodnota je znakový řetězec a jemuž je přiděleno jméno. Tímto řetězcem jsou ve skriptu nahrazeny všechny odkazy na makro.

Makro je deklarováno pomocí zvláštního elementu <xd:macro>, který musí být uveden na úrovni přímých potomků X-definice (podobně, jako modely elementů). Jméno makra je v elementu zapsáno pomocí povinného atributu "name" (resp. "xd:name"), který musí být jednoznačný v rámci celé X-definice (tj. nemohou existovat dvě deklarace makra se stejným jménem). Hodnota makra se zapisuje jako textová hodnota tohoto speciálního elementu. Odkaz na makro ("volání" makra) má ve skriptu tvar "${jméno_makra}".

Makra mohou mít také parametry (jimž je přiděleno jméno), jejichž hodnota je předdefinována v deklaraci makra pomocí atributů, které mají jméno parametru. Odkaz na parametr makra v zápisu hodnoty makra má tvar "#{jméno_parametru}". Seznam parametrů při volání makra se zapisuje v závorkách a každý parametr se uvádí jménem, jemuž je přiřazena hodnota jako znakový řetězec. Pokud parametr není při volání makra uveden, dosadí se předdefinovaná hodnota. Není-li seznam parametrů při volání makra uveden, nebo je-li prázdný, dosadí se předdefinované hodnoty. Volání makra s parametry má tedy tvar:

```
${jméno(p1 = 'hodnota 1', ...)}
```

Poznámka: Pokud má být uvnitř hodnoty parametru volání makra znak "'" nebo znak "", kterým začíná deklarace hodnoty, musí být před tímto znakem uveden znak "\".

Zpracování maker probíhá před dalším zpracováním skriptu a probíhá tak, že se volání makra nahradí jeho hodnotou. Kdekoliv se ve skriptu vyskytne odkaz na makro, je nahrazen hodnotou makra. To se děje do té doby, dokud je v textu nalezeno volání nějakého makra. V textu, který tvoří hodnotu makra, tedy může být i odkaz na jiné makro. Počet takto vnořených volání maker je omezen pevnou hodnotou 100 (zamezuje se tak "zacyklení" vnořených maker), při překročení této hranice je hlášena chyba. Volání makra může být uvedeno ve skriptu kdekoliv, i uvnitř hodnot konstant, klíčových slov, nebo identifikátorů. Z tohoto důvodu je třeba upozornit na možnost neúmyslného volání makra uvnitř deklarace znakových řetězců. V případě, že nechceme, aby byl zápis interpretován jako volání makra, je potřeba znak "$" nahradit např. zápisem "\44" (tj. zápis "\44{text}" nebude interpretován jako makro). Pokud skript X-definice obsahuje odkaz na makro v jiné X-definici, předřadí se jménu makra jméno referované X-definice, následované znakem "#" obdobně, jako v ostatních rozšířených odkazech.

Obsah makra je do skriptu zkopírován včetně mezer a nových řádek. Pomocí maker je tedy možné vložit do znakových konstant nové řádky i do skriptu, který je uveden v hodnotách atributů.

Příklad deklarací maker:

```
<xd:macro name = "jmeno">string(2,30)</xd:macro>
<xd:macro name = "barvy" p1="bílá" p2="černá">list('#{p1}',#{p2}')</xd:macro>
<xd:macro name = "prijmení">required ${jmeno}</xd:macro>
<xd:macro name = "pozdrav" p="'ahoj'">outln(#{p});<xd:macro>
<xd:macro name = "text">outln('Volání makra text má tvar:
\44{text}');"</xd:macro>
```

Příklad volání maker:

```
<osoba jmeno    = "optional ${jméno}" prijmeni = "${prijmeni}" />
<obal potisk    = "required ${barvy(p2='červená')}" />
<popiska potisk = "optional ${barvy)" />
<pozdrav xd:script = "finally ${pozdrav}">
<pozdrav xd:script = "finally ${pozdrav(p='\'nazdar\'')}">
<t xd:script = "finally ${text}">
```

Výsledek po zpracování maker preprocesorem:

```
<osoba jmeno    = "optional string(2,30)" prijmeni = "required string(2,30)" />
<obal potisk    = "required list{'bílá','červená'}" />
<popiska potisk = "optional list{'bílá','černá'}" />
<pozdrav xd:script = "finally outln('ahoj');">
<pozdrav xd:script = "finally outln('nazdar');}">
<t xd:script = "finally outln('Volání makra text má tvar:
${text}');" />
```

Poznámka: Poslední dva řádky by nebylo možné do skriptu zapsat tak, jak je uvedeno rozvinuté makro. Jednak by bylo obsaženo neúmyslné volání makra a nový řádek by byl nahražen mezerou. Možný ekvivalent zápisu by ve skriptu mohl mít např. tvar:

```
<t xd:script = "finally outln('Volání makra text má tvar:\n\44{text}');" />
```

3.2.28 Nedefinované atributy (xd:attr")

Pokud chceme, aby se ve zpracovávaných datech mohly vyskytnout i atributy, které nejsou v X-definici popsány, zapíšeme zvláštní atribut "xd:attr" a v jeho skriptu můžeme dále popsat, co se má v takovém případě stát. V následující ukázce je u elementu <a> deklarován povinný atribut "x", jehož hodnota je řetězec znaků a hodnota všech ostatních atributů musí být číslo:

```
<a x       = "required string"
   xd:attr = "occurs 0..*; int();"
/>
```

Výběr atributů dále může být podmíněn akcí "match", ve které může být výraz podmiňující selekci.

```
<a xd:attr = "match int(); occurs 1..*" >
```

Ve shora uvedené ukázce se akceptují pouze atributy, jejichž hodnota je číslo. Alespoň jeden takový atribut je povinný.

Pokud specifikujeme volbu "otherAttributes", jsou povoleny libovolné atributy a jsou zkopírovány do výsledného dokumentu.

```
<a  x = "ignored" />
<b xd:script = "option otherAttributes" />
```

V elementu <a> je povolen atribut "x", ale je ignorován (ve výsledném elementu se neobjeví). V elementu <b> jsou zpracovány libovolné atributy (jsou zkopírovány do výsledného elementu).

Poznámka: Uvedením atributu "xd:attr" říkáme, že se v daném elementu může vyskytovat libovolný počet nedefinovaných atributů, tzn., že zápis

```
<a xd:attr = "occurs 0..*; int();"/>
```

je ekvivalentní zápisu

```
<a xd:attr = "int();"/>
```

3.2.29 Nedefinované elementy ("xd:any")

Podobně jako nedefinované atributy je možné připustit v X-definici na příslušném místě výskyt elementu, jehož jméno je neznámé zápisem pomocného elementu <xd:any>. Atributy a textovou hodnotu tohoto elementu je pro takový element možné popsat běžným způsobem. Příklad zápisu libovolného elementu, který je ignorován, včetně jeho textové hodnoty:

```
<xd:any xd:script = "occurs *; forget"
        xd:attr   = "ignore" >
  ignore
</xd:any>
```

Element v uvedeném příkladě může mít libovolné jméno, může mít také libovolné atributy, které budou ignorovány, může mít i libovolnou textovou hodnotu, která bude rovněž ignorována. V uvedeném příkladě ale tento element nesmí mít žádné elementy jako potomky.

Element, který má povoleny libovolné atributy, potomky a textové uzly, je možné zapsat jako element <xd:any> a ve skriptu zapsat speciální volby "moreAttributes", "moreElements" a "moreText":

```
<xd:any xd:script = "options moreAttributes, moreElements, moreText"/>
```

Model elementu <xd:any> lze na úrovni X-definice zapsat jen, specifikujeme-li atribut "xd:name". Na tento model se pak můžeme odkazovat přes sekci "ref".

3.2.30 Ignorované textové hodnoty

Podobně jako u atributů lze ignorovat i textové hodnoty. Smysl je zřejmý z následujícího příkladu:

```
<a>
   required string()
   <b/>
   ignore
   <c/>
</a>
```

V uvedeném příkladu je požadována textová hodnota jako první prvek potomků prvku <a>, další výskyt textové hodnoty je ignorován.

3.3 Skupiny objektů

Při popisu struktur je třeba někdy popsat celé skupiny objektů. Skupiny tvoří zápis posloupnosti elementů resp. textových uzlů, včetně příslušného skriptu. Popis skupiny může být jednak umístěn přímo uvnitř modelu elementu (jako speciální pomocný element, sloužící k popisu skupiny) nebo může tvořit vlastní model.

V X-definicích jsou tři druhy skupin: neuspořádané skupiny, výběrové skupiny a uspořádané skupiny (sekvence).

3.3.1 Neuspořádaná skupina ("mixed")

Neuspořádaná skupina popisuje seznam prvků, jejichž pořadí popsaných prvků může být libovolné - neuspořádané skupině vyhovuje libovolná permutace vyjmenovaných prvků. Seznam prvků tvořících neuspořádanou skupinu se zapisuje do zvláštního elementu <xd:mixed> jako seznam potomků tohoto elementu. V neuspořádané skupině se nesmí opakovat prvek se stejným jménem (to platí i o textové hodnotě, která smí být specifikována pouze jednou). Příklad:

```
<Objekt>
   <xd:mixed>
     <Osoba   xd:script = "occurs ?" jmeno = "required string()" />
     <Firma   xd:script = "occurs ?" nazev = "required string()" />
     <Doklad  xd:script = "occurs *"  typ = "required int()" />
     optional string() /*textová hodnota */
   </xd:mixed>
</Objekt>
```

Všimněme si, že vzhledem k tomu, že ve shora uvedeném příkladu jsou všechny prvky deklarovány jako nepovinné, může být uvedená sekvence i prázdná.

Popis neuspořádané skupiny může být doplněn atributem empty="false", nebo empty="true". Implicitně je nastavena hodnota empty="true". Pokud bychom chtěli, aby skupina obsahovala alespoň jeden z uvedených prvků, zapíšeme to doplněním atributu empty="false":

```
<Objekt>
   <xd:mixed empty = "false" >
     <Osoba   xd:script = "occurs ?" jmeno = "required string()" />
     <Firma   xd:script = "occurs ?" nazev = "required string()" />
     <Doklad  xd:script = "occurs *"  typ = "required int()" />
     optional string()
   </xd:mixed>
</Objekt>
```

Takto zapsaná skupina musí obsahovat alespoň jeden prvek.

3.3.2 Výběrová skupina ("choice")

Výběrová skupina umožňuje popsat různé varianty. Jména elementů v jednotlivých variantách se musí odlišovat. Výběrová skupina se zapisuje do zvláštního elementu <xd:choice>. Příklad:

```
<Objekt>
   <xd:choice>
     <Osoba  xd:script = "occurs 1" jmeno = "required string()" />
   <Firma  xd:script = "occurs 1" nazev = "required string()" />
   </xd:choice>
</Objekt>
```

V uvedené výběrové skupině se musí vyskytnout alespoň jeden prvek. Pokud bychom chtěli připustit, že žádný z prvků skupiny nemusí vyhovovat, můžeme to vyjádřit hodnotou occurs="?", který musíme uvést do sekce skriptu u elementu <xd:choice>. Implicitně je pro výběrové skupiny nastavena hodnota occurs="1".

Výše uvedený příklad lze proto přepsat následovně:

```
<Objekt>
   <xd:choice xd:script="occurs ?">
     <Osoba    xd:script="occurs 1"   jmeno="required string()" />
     <Firma    xd:script="occurs 1"   nazev="required string()" />
   </xd:choice>
</Objekt>
```

a validní dokument může mít tvar:

```
<Objekt>
</Objekt>
```

3.3.3 Uspořádaná skupina ("sequence")

V uspořádaných skupinách je pořadí objektů shodné s jejich popisem. Pokud je sekvence uvedena uvnitř smíšené nebo výběrové skupiny, musí mít první prvek skupiny jméno, které je jednoznačné v rámci nadřazené skupiny a jeho výskyt by měl být povinný.

```
<a>
   <xd:mixed>
      <b/>
      <c/>
      <xd:sequence>
         <d/>
         required string()
         <e/>
      </xd:sequence>
   </xd:mixed>
</a>
```

V uvedeném příkladě se ve smíšené sekvenci může vyskytnout prvek <b/>, <c/> nebo posloupnost prvků <d/>, text a <e/>.

Počet opakování sekvence může být specifikováno pomocí hodnoty "occurs":

```
 <xd:sequence script="occurs 1..*">
   <d/>
   required string()
   <e/>
 </xd:sequence>
```

3.3.4 Specifikace akcí skupiny

Akce týkající se skupin se zapisují do atributů. Je možné specifikovat akce "init", "finally" a "create". Příklad:

```
<xd:sequence occurs="1..*" finally="outln('Sekvence zpracovaná')">
   <d/>
   required string()
   <e/>
 </xd:sequence>
```

Pozn.: Akce skupiny lze specifikovat dvěma způsoby:

a) uvedením každé akce jako samostatného atributu v elementu skupiny, tj. lze uvést atributy "occurs", "finally" či "create" a uvést jejich hodnoty

b) uvedením atributu "xd:script" a obvyklým způsobem vypsat příslušné akce do sekce skriptu.

Výše uvedené dvě možnosti specifikace akcí skupiny nelze kombinovat.

3.3.5 Konstrukční mód pro skupiny

Pokud je použita skupina v konstrukčním módu, je možné nastavit pro celou skupinu vstupní objekty, ze kterých je skupina sestavena. V takovém případě je třeba specifikovat v popisu skupiny atribut "create".

X-definice:

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1">
 <HTML>
    <BODY>
       <h2>create 'Seznam panovníků'</h2>
       <xd:sequence occurs="*" create="from('//panovnik')">
          create from('@jmeno');
          <br/>
       </xd:sequence>
       <h2>create 'Seznam jejich známých manželek'</h2>
       <xd:sequence occurs="*" create="from('//panovnik/manzelka')">
          create from('./text()');
          <br/>
       </xd:sequence>
    </BODY>
 </HTML>
</xd:def>
```

Vstupní data:

```
<panovnici>
```

```
   <panovnik rod="přemyslovec" jmeno="Vratislav II" />
   <panovnik rod="přemyslovec" jmeno="Přemysl Otakar I."/>
   <panovnik rod="lucemburk" jmeno="Karel IV=">
      <manzelka>Blanka z Valois</manzelka>
      <manzelka>Anna Faltská</manzelka>
      <manzelka>Anna Svídnická</manzelka>
      <manzelka>Alžběta Pomořanská</manzelka>
   </panovnik>
   <panovnik jmeno="Václav Klaus">
      <manzelka>Livie</manzelka>
   </panovnik>
</panovnici>
```

Výsledek:

```
<HTML>
   <BODY>
      <h2>Seznam panovníků</h2>
         Vratislav II<br/>
         Přemysl Otakar I.<br/>
         Karel IV.<br/>
         Václav Klaus<br/>
      <h2>Seznam jejich známých manželek</h2>
         Blanka z Valois<br/>
         Anna Faltská<br/>
         Anna Svídnická<br/>
         Alžběta Pomořanská<br/>
         Livie<br/>
   </BODY>
</HTML>
```

Příklad volání z Java kódu:

```
File xdef;
Element xmlData;
...
// 1. Create XDPool and XDDocument.
XDPool xpool = XDFactory.compileXD(System.getProperties(), xdef);
XDDocument xdoc = xpool.createXDDocument();
// 2. set source context data.
xdoc.setXDSource(xmlData);
// 3. create result
Element result = xdoc.xcreate("HTML", null);
```

3.4 Záhlaví X-definice

Záhlaví X-definice obsahuje informace, popisující její vlastnosti. Tyto vlastnosti se zapisují pomocí atributů s pevně určeným jménem kořenového uzlu X-definice, jejíž jména jsou součástí jmenného prostoru "xdef". Specifikace některých atributů je povinná a některé atributy jsou volitelné. Kromě atributů ze jmenného prostoru X-definice mohou být v záhlaví X-definice uvedeny uživatelské parametry, na jejichž hodnoty je možné se odvolávat ve skriptu uvnitř X-definice.

3.4.1 Povinné atributy v záhlaví X-definice

Záhlavím X-definice rozumíme atributy elementu <xd:def>.

3.4.1.1 Specifikace jmenného prostoru - "xmlns:xd"

```
xmlns:xd = "http://www.syntea.cz/xdef/3.1"
```

Poznámka: prefix musí být neprázdný.

Všechny speciální atributy a elementy X-definic musí mít prefix odpovídající URI identifikátoru:

```
"http://www.syntea.cz/xdef/3.1".
```

3.4.1.2 Jméno XDefinice - "xd:name"

```
xd:name = "jméno"
```

Jméno X-definice odpovídá syntaxi identifikátorů skriptu, tj. musí začínat písmenem, za nímž následuje posloupnost písmen a číslic a kdekoliv v identifikátoru se může vyskytnout znak '_'. Jméno X-definice musí být jednoznačné v rámci všech použitých X-definic, tj. nesmí být dvě různé X-definice se stejným jménem.

3.4.2 Nepovinné atributy v záhlaví X-definice

3.4.2.1 Specifikace seznamu přípustných kořenových elementů - "xd:root"

Tato specifikace se zapíše pomocí atributu "xd:root", který obsahuje seznam modelů elementů, které popisují elementy, které jsou v X-definici popsány. V jedné X-definici totiž může být popsáno více kořenových elementů, z nichž se ten odpovídající vybere podle jména. Jednotlivá jména jsou oddělena znakem "|" (čteme jako "nebo"). Představme si, že data popsaná X-definicí mohou obsahovat dva různé typy objektů: buď firmu, nebo fyzickou osobu. Příslušný seznam kořenových elementů pak může vypadat takto:

```
xd:root = "firma | osoba"
```

Jména "firma" a "osoba" jsou jména modelů elementů, které popisují kořen příchozích dat. Výběr se provede automaticky na začátku zpracování podle jména kořenového elementu dat. Ostatní případné modely elementů v X-definici mohou pak sloužit pouze jako pomocné.

Do seznamu kořenových elementů je možné přidat prvek "*", který znamená, že X-definici vyhovuje jakýkoliv element. Zápis "firma | osoba | *" znamená, že kořenem zpracovaných dat může být element "firma" nebo "osoba" nebo jakýkoliv jiný element.

3.4.2.2 Skript - "xd:script"

Skript X-definice může obsahovat deklaraci skriptu, ve kterém je povolena pouze specifikace akcí "init", "onIllegalRoot", "onXmlError" a specifikace voleb ("options"). V okamžiku, kdy je nalezen v příslušné X-definici odpovídající kořenový model elementu, provede se akce "init", není-li nalezen, provede se akce "onIllegalRoot". Seznam voleb, povolených ve skriptu X-definice, smí obsahovat pouze následující volby:

noSetAttrCase	setAttrLowerCase	setAttrUpperCase
noSetTextCase	setTextLowerCase	setTextUpperCase
ignoreAttrWhiteSpaces	preserveAttrWhiteSpaces	acceptEmptyAttributes
ignoreTextWhiteSpaces	preserveTextWhiteSpaces	
noTrimAttr	trimAttr	
noTrimText	trimText	
ignoreComments	preserveComments	
ignoreEmptyAttributes	preserveEmptyAttributes	

Nejsou-li volby uvedeny, pak přednastavené (default) hodnoty jsou:

```
noSetAttrCase
noSetTextCase
preserveAttrWhiteSpaces
preserveTextWhiteSpaces
trimAttr
trimText
ignoreComments
preserveEmptyAttributes
```

3.4.2.3 Odkaz na další X-definice - "xd:include"

Do atributu "xd:include" je možné zapsat seznam odkazů URL nebo cestu k souborům na další X-definice nebo kolekce. Oddělovačem v seznamu je čárka. Pokud je odkaz na soubor v lokálním systému, může být uveden i bez prefixu "file:" a pak se jeho umístění interpretuje jako relativní cesta vzhledem k umístění souboru, ve kterém je odkaz zapsán. Ukazuje-li odkaz na soubor, je možné v zápisu použít i znaky pro označení skupiny souborů, tj. místo znaků ve jméně souboru je možné uvést znak "*", případně "?" (např. zápis "*.xdef" znamená všechny soubory s příponou "xdef", zápis "a??.def" odkazuje na všechny soubory, které mají jméno začínající písmenem "a" za nímž následují dva libovolné znaky a mají příponu "def"). Není-li odkazovaný objekt lokální soubor, nelze ve jménu použít znaky "*" a "?".

Odkaz může být také na objekt, který je přístupný pomocí URL (pak musí být uveden protokol, přes který je soubor přístupný - např. "http://"). Objekt, na který ukazují tyto odkazy, mohou být jak X-definice, tak i odkazy na kolekce. V odkazovaných souborech samozřejmě mohou být opět další odkazy.

Příklad:

```
<xd:def xmlns:xd   = "http://www.syntea.cz/xdef"
        xd:name    = "dummy"
```

```
        xd:include = "http://www.syntea.cz/project/xdef1.def"
        xd:script  = "onIllegalRoot outln(‘Illegal root‘)" >
```

3.4.2.4 Definice jmenného prostoru pro objekty X-definic - "xd:metaNamespace"

Aby bylo možné X-definicemi popsat i samotné X-definice, je možné zadat atribut "xd:metaNamespace" ze jmenného prostoru X-definic, kterým je možné specifikovat namespace URI, který bude nadále interpretován jako namespace pro objekty X-definic. Navíc je možné se na tyto objekty odkazovat.

Příklad:

```
<meta:def xmlns:meta       ="http://www.syntea.cz/project/xdef"
          xmlns:xd         ="http://www.syntea.cz/xdef"
          name             ="dummy"
          xd:metaNamespace ="http://www.syntea.cz/project/xdef" >

  <xd:def xd:name          ="required string()">
    <meta:any meta:script="?; options moreAttributes, moreElements, moreText"/>
  </xd:def>
</meta:def>
```

3.4.2.5 Implementační atributy - "impl-"

Někdy je vhodné mít k dispozici také hodnoty, které slouží jako implementační parametry či data pro určitou aplikaci. K tomu slouží atributy, které můžeme do záhlaví X-definice přidávat sami dle uvážení, jejichž jméno začíná posloupností znaků "impl-". Např. atribut "impl-verze" může nést informaci o verzi implementace. Ve skriptu lze hodnotu implementačních atributů získat pomocí metody getImplProperty(jmeno). Jméno se zapisuje bez prefixu "impl-", tedy např. getImplProperty('verze') pro atribut "impl-verze".

4 Kolekce X-definic

V praxi obvykle nelze pro složitější aplikace vystačit s jedinou X-definicí. Více X-definic je možné vložit do objektu, který nazýváme **kolekce** X-definic, který slouží jako "obálka" více X-definic. X-definice vkládáme do kolekce jako potomky kořenového elementu <xd:collection>. X-definice jsou tedy do elementu <xd:collection> vnořené jako elementy. Kolekce rovněž umožňuje (obdobně jako X-definice) uvést odkaz na další X-definice nebo kolekce pomocí speciálního atributu "xd:include". Jeho formát i význam je shodný s tímto atributem v X-definici (viz odstavec 3.4.2.3). Jiné atributy, než "xd:include" nejsou v kolekci povoleny.

Příklad využití elementu <xd:collection>:

```
<xd:collection xmlns:xd = "http://www.syntea.cz/xdef/3.1">
        <xd:def xd:name  = "Osoba" xd:root  = "Osoba" >
                /* Obsah X-definice Osoba */
        </xd:def>

        <xd:def xd:name  = "Firma" xd:root  = "Firma" >
                /* Obsah X-definice Firma. */
        </xd:def>
</xd:collection>
```

V kolekci může být také atribut "include" (jako v X-definici). Tímto způsobem můžeme sestavit XDPool. Například:

```
<xd:collection xmlns:xd   = "http://www.syntea.cz/xdef/3.1"
               xd:include = "http://www.syntea.cz/project/xdef*.def, C:/project/mydef.xd"/>
```

5 Použití BNF v X-definicích

5.1 BNF gramatika

Gramatika je popsána pomocí rozšířené Backus Naurovy formy (Extended Backus Naur Form - EBNF), která popisuje formální syntaxi pomocí produkčních pravidel.

5.1.1 Produkční pravidla

Každé produkční pravidlo má jméno. Jméno pravidla musí začínat písmenem nebo znakem "_" (podtržítko), a následovat může posloupnost číslic, písmen nebo znaku "_" (podtržítko). V zápisu EBNF je jméno pravidla nalevo od znaků "::=", na pravé straně následuje výraz, popisující pravidlo. Každé syntaktické pravidlo definuje jeden symbol gramatiky formou:

```
symbol ::= výraz
```

kde na levé straně operátoru "::=" je jméno pravidla a na pravé straně je popsáno pravidlo pomocí BNF výrazu.

5.1.2 Jednoduché (terminální) symboly

Jednoduché (terminální) symboly jsou popsány pomocí následujících konstrukcí:

#xN	odpovídá znaku, jehož číselná hodnota (kód) je N, kde N je hexadecimální číslo. Vedoucí nuly jsou ignorovány. Kódová tabulka je ISO/IEC 10646.
"string" nebo **'string'**	odpovídá řetězci znaků v uvozovkách nebo apostrofech.

5.1.3 Množiny znaků

[a-zA-Z] nebo **[#xN-#xN]**	odpovídá znakům, jejichž hodnota je v uvedeném uzavřeném intervalu.
[abc] nebo **[#xN#xN#xN]**	odpovídá znakům, jejichž hodnota je v uvedeném výčtu. V hranatých závorkách mohou být uvedeny v libovolném pořadí zápisy intervalu i výčtu.
[^a-z] nebo **[^#xN-#xN]**	odpovídá všem znakům, které leží mimo uvedený interval.
[^abc] nebo **[^#xN#xN#xN]**	odpovídá všem znakům, které nenáleží do uvedeného výčtu. V hranatých závorkách mohou být uvedeny v libovolném pořadí zápisy jak intervalu, tak i výčtu.

5.1.4 Specifikace opakování komponenty (kvantifikátory)

Kvantifikátory umožňují popsat povolený počet po sobě následujících výskytů řetězce odpovídajícím pravidlu, na nějž se kvantifikátor vztahuje:

A?	odpovídá řetězci znaků, který splňuje pravidlo A nebo ničemu (tj. A je volitelné, tj. nula nebo jeden výskyt).
A+	odpovídá jednomu nebo více po sobě následujícím výskytům řetězce znaků splňujícím pravidlo A.
A*	odpovídá žádnému nebo více po sobě následujícím výskytům řetězce znaků splňujícím pravidlo A.
A{n}	odpovídá n po sobě následujícím výskytům řetězce znaků splňujícím pravidlo A.
A{m,n}	odpovídá minimálně m a maximálně n po sobě následujícím výskytům řetězce znaků splňujícím pravidlo A.
A{m,}	odpovídá minimálně m a dále neomezeně po sobě následujícím výskytům řetězce znaků splňujícím pravidlo A.

5.1.5 BNF výrazy

Shora uvedené konstrukce mohou být uvedeny ve složených pravidlech, popisující složitější neterminální symboly. Výrazy na pravé straně mohou obsahovat uvedené prvky nebo odkazy na jiné pravidlo pomocí jména pravidla a mohou být složeny z následujících komponent. Jednotlivé části zápisu mohou být v závorkách:

A - B	odpovídá řetězci znaků, který splňuje pravidlo A, ale nesplňuje pravidlo B (omezení). Omezení má vyšší prioritu, než spojení (nebo alternativa), tedy: **A - B C - D** je identické se zápisem: **(A - B) (C – D)** a také: **A - B | C - D** je identické se zápisem: **(A - B) | (C - D)**
A B	odpovídá řetězci znaků podle pravidla A, který je následován B (sekvence). Spojení má vyšší prioritu, než alternativa, tedy: **A B | C D** je identické se zápisem **(A B) | (C D)**

A | B odpovídá řetězci znaků podle pravidla A nebo řetězci podle pravidla B (alternativa).

5.1.6 Externí pravidla

V BNF gramatice je možné odvolat se na externí pravidla, která mohou být v externích Java programech. Externí pravidlo musí začínat znakem "$", za nímž následuje jméno externí metody. Jméno externí metody může obsahovat plně kvalifikovanou cestu (tj. jméno package, třídy a metody) nebo to může být odvolávka na alias jméno.

5.1.7 Definiční sekce - jména a parametry pro externí metody

Externí metody je možné definovat pomocí příkazu "%define" za nímž následuje krátké alias jméno, následované dvojtečkou a jméno externí metody. Pokud má externí metoda parametry, mohou být jejich hodnoty popsány v závorkách. Parametry jsou odděleny čárkou. Hodnoty mohou být typu integer, float, string, datum, duration. Tyto definice musí být uvedeny před dalším popisem gramatiky.

Příklad:

```
%define $pravidlo1: $dev.myproject.BNFPravidla.pravidlo1
%define $pravidlo2: $dev.myproject.BNFPravidla.pravidlo(123, "abc")
%define $pravidlo3: $dev.myproject.BNFPravidla.pravidlo(-1)
%define $operator: $pushParsedObject("op")
%define $datum: $datetime("dd.MM.yyyy")

Pravidlo1 ::= $pravidlo1 | $pravidlo2 | $pravidlo3
```

5.1.8 Poznámky a mezery v zápisu BNF gramatiky

Poznámka je text umístěný mezi znakovými řetězci "/*" a "*/". Vnoření poznámek není dovoleno.

`/* ... */` `text, který neobsahuje sekvenci znaků "*/".`

Poznámka je procesorem BNF ignorována.

Kdekoliv v textu, mimo zápis terminálních symbolů a jmen pravidel, mohou být libovolné sekvence mezer, nových řádků a tabelátorů a poznámek.

5.1.9 Předdefinované metody

Interpret BNF gramatiky má předdefinovaná následující pravidla:

`$integer`	celá čísla
`$float`	čísla s pohyblivou řádovou čárkou
`$decimal`	decimální čísla
`$digit`	číslice
`$letter`	písmeno
`$lowercaseLetter`	malé písmeno
`$uppercaseLetter`	velké písmeno
`$letterOrDigit`	písmeno nebo číslice
`$boolean`	"true" nebo "false"
`$datetime`	datum a čas podle specifikace ISO (musí být v definiční sekci s parametrem formátu)
`$date`	datum podle specifikace ISO
`$time`	čas podle specifikace ISO
`$yearMonth`	rok a měsíc podle specifikace ISO
`$monthDay`	měsíc a den podle specifikace ISO
`$month`	měsíc podle specifikace ISO
`$day`	den podle specifikace ISO
`$year`	rok podle specifikace ISO
`$duration`	časový interval podle specifikace ISO
`$base64`	data v base64 formátu
`$hexData`	data v hexadecimálním formátu

$xmlName	XML jméno podle specifikace W3C
$ncnNme	NCNAME podle specifikace W3C
$nmToken	NMTOKEN podle specifikace W3C
$xmlChar	validní XML znak podle specifikace W3C
$whitespace	whitespace znak podle specifikace W3C
$xmlNamestartchar	první písmeno jména podle specifikace W3C
$xmlNameExtchar	následující písmena jména podle specifikace W3C
$clearParsedObjects	vymazání zásobníku parsovaných objektů
$pushParsedObject	uložení hodnoty z parametru do zásobníku parsovaných objektů (parametr musí být uveden v definiční sekci)
$pushParsedString	uložení parsovaného stringu do zásobníku parsovaných objektů
$popParsedObject	zrušení položy na vrcholu zásobníku parsovaných objektů
$anyChar	přeskočí jakýkoliv znak a vrátí true, na konci zdrojového textu vrátí false
$find	přeskočí source až do místa, které odpovídá argumentu. Vrátí true, je-li řetězec nalezen
$findOneOfChars	přeskočí source až do místa, které odpovídá jednomu ze znaků v argumentu. Vrátí true, je-li znak nalezen
$error	zapíše chybovou hlášku a vrátí false.

5.2 Použití BNF v X-definici

Objekt obsahující přeloženou BNF gramatiku můžeme vytvořit pomocí elementu na úrovni X-definice (podobně jako deklarační sekci):

```
<xd:BNFGrammar name = "jmeno">
   Popis pravidel gramatiky – viz předchozí popis
</xd:BNFGrammar>
```

Z objektu jedné gramatiky lze vytvořit novou gramatiku rozšířenou o další pravidla pomocí deklarace s atributem "extends":

```
<xd:BNFGrammar name = "jméno" extends = "jméno_základní_gramatiky" >
   Popis dalších pravidel rozšířené gramatiky
</xd:BNFGrammar>
```

Tento zápis je především přehledný a snadno čitelný. BNF gramatiku můžeme ale také vytvořit pomocí konstruktoru objektu BNFGrammar:

```
<xd:declaration xd:scope="global">
  BNFGrammar g = new BNFGrammar("numbers ::= [0-9]+ ( ',' [0-9]+ )*");
        …
  /* g1 je g rozsirena o pravidlo „hexa“ */
  BNFGrammar g1 = new BNFGrammar("hexa ::= ('X' | 'x') [0-9A-F]+", g);

</xd:declaration>
```

Pravidlo z gramatiky získáme pomocí metody "rule" implementované v objektech typu BNFGrammar. Například:

```
   BNFRule x = g1.rule("hexa");
```

Je-li pravidlo uvedeno v sekci typové kontroly atributu nebo textové hodnoty, provede se automaticky kontrola textu podle tohoto pravidla:

```
   <elem a="optional x">
     required g1.rule("numbers");
   </elem>
```

Atribut "a" elementu <elem> musí vyhovovat pravidlu "hexa" z gramatiky g1 a textová hodnota tohoto elementu pravidlu "numbers".

V průběhu typové kontroly se automaticky vyvolá metoda "Cheb" s textovou hodnotou atributu nebo elementu.

Kontrolu libovolného stringu pomocí určitého pravidla můžeme provést pomocí metody "check", která je implementována v objektech typu BNFRule. Výsledkem je boolean hodnota. Je-li "true", pak string odpovídá příslušnému pravidlu. Například v příkladu:

```
BNFGramar g = new BNFGramar("A ::= [0-9]+(',' [0-9]+)*")
BNFRule r = g.rule("A");
String s = "123,4,5";
boolean x = r.check(s);
```

bude mít proměnná "x" hodnotu "true".

6 Činnost procesoru X-definic

Existují dva zásadně odlišné režimy práce procesoru X-definic. První režim, který nazýváme "**režim validace**" zpracovává vstupní data a kontroluje výskyt objektů podle X-definice. Proces je v tomto režimu řízen zpracováním vstupního zdrojového tvaru XML objektu, k němuž se vyhledají odpovídající části X-definice. Druhý režim, kterému říkáme "**režim konstrukce**" je řízen X-definicí (nikoliv vstupními daty jako v prvním případě). Proces je tomto případě řízen zpracovávanou X-definicí, která slouží jako předpis, podle kterého se tvoří výsledek. První režim je spíše vhodný ke kontrole dat, druhý k transformacím vstupních objektů do jiného tvaru nebo k jejich vytváření.

6.1 Režim validace (kontroly dat)

V tomto režimu je průběh zpracování řízen vstupními daty dokumentu. Pokud jsou vstupní data ve formě zdrojového XML dokumentu, kontrola probíhá sekvenčně a akce se provádějí v okamžiku, kdy syntakticky řízený procesor zpracovává jednotlivé části XML dokumentu. Pokud jsou vstupní data zadána jako hotový objekt "org.w3c.dom.Element", je pořadí akcí dáno rekurzívním průchodem stromu dokumentu. V obou případech je zpracování obdobné. Situace, které v průběhu zpracování popisujeme X-definicí, jsou následující:

1. **Začátek zpracování kořenového elementu**. V tomto okamžiku je známo pouze jméno kořenového elementu XML dokumentu a dochází k výběru příslušné definice. Události, které mohou nastat, jsou **init** (volá se před dalším zpracováním) a **onIllegalRoot** (v X-definici nebyl nalezen model elementu, jehož jméno odpovídá seznamu kořenových elementů - viz "xd:root"). Tyto události lze popsat v záhlaví X-definice do atributu "xd:script". V tomto okamžiku je k dispozici příkazům skriptu pouze jméno kořenového elementu a jeho atributy a instance objektu "org.w3c.dom.Document", která je dále procesorem používána k vytváření stromu XML dokumentu. Podle nalezené X-definice pak pokračuje další zpracování. Není-li X-definice nalezena a není popsána událost **onIllegalRoot**, procesor ohlásí chybu a ukončí práci.
2. **Začátek elementu**. V tomto okamžiku je vytvořen nový element daného jména a je připojen jako potomek nadřazeného objektu (tj. dokumentu nebo elementu). Pokud je popis nalezen, nastane událost **init** a provede se kontrola, zda výskyt elementu nepřekračuje maximální povolený počet. Pokud ano, nastává událost **onExcess** (nový element je nadbytečný) a provede se příslušná akce skriptu (pokud není uvedena, ohlásí se chyba). Jestliže popis elementu není v X-definici nalezen, nebo když je tento element popsán jako "illegal", nastane událost **onIllegalElement**. V případě, že je tato událost popsána ve skriptu, provede se příslušná akce skriptu, jinak se ohlásí chyba. V této situaci jsou k dispozici všechny atributy elementu (v ještě nezpracované formě).
3. **Zpracování seznamu atributů**. Následuje krok, ve kterém se prochází seznam atributů. Při výskytu jednotlivých atributů se nejdříve zkontroluje, zda je atribut podle definice přípustný. Není-li atribut v definici, nebo je-li deklarován jako "illegal", vyvolá se akce **onIllegalAttr**. Není-li deklarována, ohlásí se chyba. Následuje vyvolání kódu pro validaci obsahu. Je-li výsledek validace "true", nastává událost **onTrue** a atribut se připojí do výsledného elementu (pokud není v kódu akce **onTrue**, atribut je odstraněn??). V opačném případě nastává událost **onFalse**. Pokud není popsána akce **onFalse**, ohlásí se chyba a atribut se do výsledného dokumentu nepřipojí. Pokud je atribut popsán v X-definici, ale v datech chybí, nastane událost **onAbsence**. Je-li popsána akce **onAbsence**, může např. nastavit hodnotu atributu a zpracování pak pokračuje stejně, jako kdyby byl atribut ve vstupních datech - doporučujeme vymazat v tomto případě chybu pomocí metody clearReports, jinak se chyba ohlásí. Pokud akce není popsána, ohlásí se chyba.
4. **Zpracování obsahu elementu**. Po zpracování začátku elementu a seznamu atributů nastává událost, pojmenovaná **onStartElement**. V tomto okamžiku je již k dispozici vzniklý element po kontrole a zpracování atributů, ale ještě bez potomků. Ve vyvolané akci je možné např. zpracovat vztahy mezi atributy a rozpracovaným stromem celého dokumentu. Upozornění: tato událost nastává i v případě, že element nemá žádného potomka ani textovou hodnotu.
5. **Výskyt elementu**. Existuje-li element ve vstupních datech jako potomek pokračuje se krokem 2.

6. **Výskyt textové hodnoty**. Výskyt textové hodnoty elementu se zpracovává podobným způsobem, jako výskyt nového atributu. Je-li textová hodnota popsána jako "illegal", nastává událost **onIllegalText**. (místo "onIllegalAttr" u atributů). Ostatní události a příslušné akce jsou stejné jako u atributů. Upozornění: Textová hodnota je systému předána až po vyřešení odkazů na případné entity. Pokud následuje ve vstupních datech po sobě více textových hodnot, jsou tyto hodnoty spojeny v jednu hodnotu, což se týká i CDATA sekcí.
7. **Ukončení zpracování elementu**. Po zpracování vnitřních uzlů elementu se zkontroluje, zda výsledek zpracování obsahuje v požadovaném minimálním počtu vnitřní elementy a textové hodnoty. Pokud ne, nastává událost **onAbsence**, popsaná u příslušné definice elementu (není-li popsána, hlásí se chyba). Po kontrole obsahu elementu nastává událost **finally**. Příkazy pro akci **finally** se provedou nejdříve pro všechny atributy a nakonec se provede tato akce, je-li specifikována ve skriptu elementu. Na závěr, je-li zapsána akce **forget**, se obsah elementu odstraní z paměti a odstraní se i z nadřazeného uzlu XML stromu. Příslušné čítače počtu výskytu však zůstanou nastaveny.
8. Je-li zpracováván XML dokument ve zdrojovém tvaru, může nastat událost **onXmlError**. Akce pro obsloužení této události může být specifikována ve skriptu X-definice. Je-li v X-definici zapsána, lze ve skriptu rozhodnout, jak chybu obhospodařit. Zpracování je pak ukončeno, pokud se jedná o fatální chybu, nebo je záznam o chybě zapsán do protokolu chyb.

Po skončení zpracování kořenového elementu se vyvolá akce **finally** popsaná v skriptu příslušné X-definice (je-li popsána) a procesor ukončí činnost. Pokud je procesor vyvolán z programu, na výstupu odevzdá vytvořený dokument (nebyla-li uvedena akce **forget**).

6.2 Režim konstrukce (tvorby XML objektů)

V režimu konstrukce je proces řízen X-definicí. Místo zpracování a kontroly vstupních dat se vytváří výsledný objekt podle návodu, obsaženého v X-definici (např. v akcích **create**). Vstupní objekt může mít zcela jinou strukturu, nebo dokonce vůbec nemusí existovat. Pokud existuje, jsou vstupní data předána procesoru jako objekt ve tvaru XML elementu. Pomocí akcí **create** pak lze popsat data, použitá k tvorbě výstupního objektu. X-definice v tomto případě popisuje, jak má vypadat výsledná struktura XML objektu. Aby nebylo nutné popisovat akci **create** i u objektů, jejichž struktura je shodná s výstupním objektem, není nutné ji psát tam, kde jsou odpovídající údaje se stejným jménem, typem a pořadím obsaženy ve vstupním objektu. V režimu konstrukce by neměly pro správnou X-definici nastat událost **onExcess**, **onIllegalElement**, **onIllegalAttr** a **onIllegalText**. Postup zpracování je následující:

1. **Výběr modelu elementu**. V X-definici se najde buď podle jména kořenového elementu dat, nebo podle parametru, zadaného při volání programu (viz metodu XDDocument.xcreate), podle něhož se zahájí zpracování. Není-li kořenový model elementu nalezen, nastává událost **onIllegalRoot**, provede se odpovídající akce a ukončí se činnost. Jinak se postupně vybere odpovídající model elementu a pokračuje se krokem 2.
2. Provede se akce **create** podle skriptu kořenového modelu elementu. Není-li akce popsána, hledá se element odpovídajícího jména ve vstupních datech (je-li takových elementů ve vstupních datech více, než je povolený maximální počet výskytů, vybere se jen odpovídající počet). Jinak nastávají stejné události jako v režimu validace v kroku 2 (kromě události **onExcess**, která v tomto režimu nenastane).
3. Provedou se akce **create** popsané v seznamu atributů definice. Po takto vytvořeném seznamu atributů zpracování pokračuje stejným způsobem jako v režimu validace v kroku 3.
4. Provedou se akce **create** pro všechny potomky definice elementu. Poté následuje zpracování jako v krocích 4 až 6 v režimu validace.
5. Následuje zpracování podobné jako v kroku 7 v režimu validace.

7 Specifikace X-definic v XML souboru

X-definici pro zpracování dat je možné také specifikovat přímo v datovém souboru pomocí atributů ve zpracovávaném XML dokumentu, pomocí atributů ze jmenného prostoru:

```
xmlns:xdi = "http://www.syntea.cz/xdef/instance"
```

Pomocí atributu "xdi:location" lze pak uvést URL odkaz na X-definici nebo na kolekci X-definic. Pokud je odkaz na soubor v lokálním systému, může být uveden bez prefixu "file://" a pak se jeho umístění uvádí relativně k datovému

objektu jako relativní cesta ke jménu souboru s X-definicí nebo ke kolekci X-definic. V parametru je možné zapsat i seznam X-definic či souborů s objekty "xdi:location", oddělovačem je znak "," (čárka). Z takto definované množiny X-definic je pak možné pomocí atributu "xdi:xdefName" specifikovat jméno X-definice, která popisuje příslušný datový objekt.

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

<data xmlns:xdi       = "http://www.cz.syntea/xdef/instance"
      xdi:location    = "file://C:/xdef/mojeData.xdef"
      xdi:xdefName    = "mojeData"
...

</data>
```

8 Způsob tvorby X-definice podle XML dat

V této kapitole popíšeme, jak je možné vytvořit X-definici z XML dokumentu.

a) Z popisovaného XML dokumentu vytvoříme nejprve popis struktury. Nezapomeňme, že musíme popsat i ty datové struktury, které se v konkrétní instanci dat nevyskytují. Pokud (zatím) nějaká hodnota nemá pro zpracování význam (např. se s ní počítá do budoucnosti), můžeme výskyt ve skriptu popsat klíčovým slovem "ignore". Pokud se mohou vyskytnout i atributy, které neznáme, můžeme popsat tuto situaci atributem "xd:attr".

b) Popíšeme kontrolu typů hodnoty atributů a textových uzlů. Pokud chceme dále aktivovat nějaké operace v závislosti na výsledku kontrol, za kontrolní výraz popíšeme akce "onTrue" a "onFalse" a případně popíšeme akce pro různé události.

   Požadujeme-li u některého atributu nebo textového uzlu pevnou hodnotu, můžeme ve skriptu použít zápis "fixed":

   ```
   Verze = "fixed '2.0'"
   ```

   který je ekvivalentní zápisu:

   ```
   Verze = "required eq('2.0') onAbsence setText('2.0')"
   ```

c) Jestliže chceme připustit i jiné elementy, než byly popsány, přidáme do skriptu element <xd:any> (viz odstavec 3.2.29).

d) Opakující se a podobné vnitřní části elementů nahradíme vytvořením pomocných modelů elementů a příslušnými odkazy. Referované modely elementů mohou být zapsány i v různých X-definicích. Například:

   ```
   <xd:def xmlns:xd = "http://www.syntea.cz/xdef/3.1"
           xd:name = "Osoba"
           xd:root = "Osoba" >

     <Osoba jmeno = "required string()" prijmeni = "required string()" >
       <Pobyt xd:script = "ref Firma#Adresa"
              druh = "required list('trvala','docasna')"/>
     </Osoba>
   </xd:def>

   <xd:def xmlns:xd = "http://www.syntea.cz/xdef/3.1"
           xd:name = "Firma"
           xd:root = "Firma" >

     <Firma jmeno = "required string()" prijmeni = "required string()" >
       <Adresa xd:script = "ref Adresa" />
     </Firma>

     <Adresa ulice = "required string()"
             cp    = "optional int()"
             mesto = "required string()"
             PSC   = "optional num(5)"
             stat  = "required string(2)" />

   </xd:def>
   ```

   V uvedeném příkladě je struktura elementu <Pobyt> uvnitř elementu <Osoba> popsána v X-definici <Firma> modelem elementu <Adresa>, ale navíc musí obsahovat povinný atribut "druh".

e) Pokud nemusí být pořadí elementů v rámci nadřazeného elementu dodrženo, je možné popis těchto částí vložit do smíšené skupiny v pomocném elementu <xd:mixed>. Příklad:

```
  <Manzelstvi Uzavreno = "required datetime('d.M.y')">
     <xd:mixed>
        <Manzel   xd:script = "ref Osoba#Osoba" />
        <Manzelka xd:script = "ref Osoba#Osoba" />
     </xd:mixed>
  </Manzelstvi>
```

f) Výběr jedné z více variant se zapíše pomocí výběrové sekce, ohraničené pomocným elementem <xd:choice>:

```
  <Vozidlo>
     <xd:choice>
        <auto SPZ       = "required string(7,10)"
              Vyrobce   = "required string() "
              PocetMist = "required int()"/>
        <kolo Znacka    = "required string()"/>
     </xd:choice>
  </Vozidlo>
```

Poznamenejme, že výběrové sekce mohou být vnořeny do smíšené sekce.

g) Pokud chceme v jedné X-definici popsat více přípustných XML dokumentů, je třeba pomocí atributu "xd:root" popsat, které elementy se mohou vyskytnout jako kořenové elementy. Příklad:

```
<xd:def xmlns:xd = "http://www.syntea.cz/xdef/3.1"
        xd:name  = "dokument"
        xd:root  = "Osoba | Firma | Manzelsvi" >

   <Osoba Jmeno    = "required string(1,24)"
          Prijmeni = "required string(1,30)"
          RC       = "required num(10,11)"
          Pohlavi  = "required list('muz','zena')"
          Telefon  = "optional num(6,12)" >

      <Pobyt xd:script = "ref Adresa occurs 1..2"
             Druh = "required list('trvala','docasna')" />

   </Osoba>

   <Firma NazevFirmy = "required string()"
          IC         = "required num(8)"
          Telefon    = "optional num(6,12)" >

      <Adresa xd:script = "ref Adresa occurs 1" />

   </Firma>

  <Manzelstvi Uzavreno = "required datetime('d.M.y')">
     <xd:mixed>
       <Manzel   xd:script = "ref Osoba" />
       <Manzelka xd:script = "ref Osoba" />
       <Dite     xd:script = "*; ref Osoba" >
     </xd:mixed>
  </Manzelstvi>

   <Adresa Ulice           = "? string(2,30)"
           CisloOrientacni = "? int(1,99999)"
           CisloPopisne    = "int(1,99999)"
           Obec            = "string(2,30)"
           Okres           = "string(2,30)"
           PSC             = "? int(10000,99999)" />

</xd:def>
```

V souboru X-definic je možné odkazovat se na všechny X-definice daného souboru. Pokud se jedná o odkaz na jinou X-definici v rámci souboru, zapíšeme takový odkaz tak, že uvedeme jméno X-definice následované znakem '#', za nímž následuje jméno elementu definice. Například:

```
<xd:def xmlns:xd = "http://syntea.cz/xdef/3.1" xd:name = "glob"
        xd:root = "dokument#Firma | dokument#Osoba | Text " >

   <Text> required string() </Text>
</xd:def>
```

Všimněme si, že první dva prvky seznamu dokumentů odkazují na popis elementů <Firma> a <Osoba> v X-definici "dokument" a poslední pak odkazuje na popis elementu <Text> v rámci X-definice "glob".

h) Pokud to aplikace vyžaduje, je možné popsat globální proměnné, a uživatelské metody pro kontrolu typů a případně pro často používané akce. Příklad:

```
<xd:variable>
  dnes = now(); /*globalni promenna s dnesnim datem*/
</xd:variable>

<xd:declaration xd:scope="global">
/* Kontrola datového typu rodného čísla */
boolean rodneCislo() {
   if (!string(10,11)) {
      return error('Nespravna delka');
   }
   s = cut(getText(),6); /* prvnich 6 cislic do s */
   if (!num(s) {
      return error('Nenumericka hodnota');
   }
         den = parseInt(cut(s,2)); /* den narozeni jsou prvni dva znaky */
   if (den < 1 || den > 31) {
      error('Nespravny den narozeni');
   }
   mesic = parseInt(substring(s,2,4)); /* mesic dalsi dva znaky */
   if ((mesic < 1 | mesic > 12) & (mesic < 51 | mesic > 62)) {
      return error('Nespravny mesic narozeni');
   }
   rok = parseInt(substring(s,4,6)) /*rok dalsi dva*/
   /* rok nekontrolujeme, muze byt 0 az 9 */
   if (substring(getText(),6) != '/') {
      return error('Na seste pozici neni lomitko');
   }
   s = tail(getText(),7); /* identifikacni cislo je zbytek hodnoty */
   if (!num(s)) {
      return error('Nenumericka hodnota');
   }
   /* kontrolni soucet */
   n =  parseInt(s); /* prevedeme na cislo */
   if (rok * 10000000 + mesic * 100000+den * 1000 + (n/10) % 11 != n % 10) {
      return error('Chyba kontrolniho souctu');
   }
   return true; /* Rodne cislo je OK */
}

/* vystup textove hodnoty jako datum a cas */
void writeDateTime() {
   out(parseDate(getText()),"'datum= 'dd/MM/yyyy' cas= ' HH:mm:ss");
}

]]>
</xd:declaration>
```

i) Při zpracování velkých souborů se zdrojovým textem XML je možné výkonnému aparátu sdělit, že po skončení zpracování elementu je možno jej "zapomenout", aby zbytečně neobsazoval operační paměť. Tuto skutečnost zapíšeme jako akci "forget".

9 Možnosti nastavení vlastností zpracování

Některé vlastnosti zpracování je možné nastavit pomocí systémových "properties" nebo nastavením parametru properties při vytvoření objektu XDPool (je-li tento parametr "null", použijí se systémové Properies). Hodnoty properties jsou deklarovány jako konstanty ve třídě **cz.syntea.xdef.XDConstants** následovně:

Zpracování XML DOCTYPE (XDConstants.XDPROPERTY_DOCTYPE)

"xdef.doctype" "true" (default) "false"

Zpracování XML include (XDConstants.XDPROPERTY_XINCLUDE)

"xdef.xinclude" "true" "false" (default)

Čtení systémových proměnných (XDConstants.XDPROPERTY_ENV_GET)

"xdef.envGet"	"true" (default)	"false"	

Generování varovných hlášení (XDConstants.XDPROPERTY_WARNINGS)

"xdef.warnings"	"true" (default)	"false"	

Interface externích metod (XDConstants.XDPROPERTY_EXTERNALMODE)

"xdef.externalMode"	"new"(default)	"old"	"both"

Nastavení ladicího režimu (XDConstants.XDPROPERTY_DEBUG)

"xdef.debug"	"true"	"false" (default)

Nastavení výstupu při ladění (XDConstants.XDPROPERTY_DEBUG_OUT)

"xdef.debug.out"	cesta k souboru, kam je výstup uložen (default je stdOut).

Nastavení vstupu při ladění (XDConstants.XDPROPERTY_DEBUG_IN)

"xdef.debug.in"	cesta k souboru, kde jsou příkazy k ladění (default je stdIn)

Nastavení řežimu zobrazení zdrojového kódu (XDConstants.XDPROPERTY_ DISPLAY)

"xdef.display"	"true”	zobrazuje kód vždy
	"errors”	zobrazuje jen při chybě
	"false” (default)	nezobrazuje nikdy

10 Spuštění režimu validace z příkazové řádky

Program je možné spustit z příkazové řádky Java třídy "cz.syntea.xdef.util.XValidate". Ostatní použité programové prostředky jsou součástí běžného vybavení java. Pokud jsou ve skriptu použity uživatelské metody, musí být v classpath přístupné třídy, ve kterých jsou příslušné metody. Parametry při spuštění z příkazové řádky mají běžný formát:

```
[-h] [-d defList] [-x xDefName] [-l logFile] –i xmlFile
```

Význam parametrů:

-h Help (vytiskne soubor s informací o parametrech spuštění). Ostatní parametry jsou v tomto případě ignorovány.

-d defList Parametrem se popíše cesta k souboru X-definic (nebo seznamu souborů, oddělených středníkem). V zápisu jména souboru je možné použít znaky "*" a "?", pomocí kterých je možné popsat výběr skupiny souborů. Parametr je nepovinný a pokud není uveden, očekává se, že přístup k X-definici bude popsán přímo v XML dokumentu pomocí atributu "xdi:location".

-x xDefName Jméno X-definice, která má být použita pro vstupní data. Parametr je nepovinný a uvádí se jen tehdy, je-li uveden parametr "-d".

-l logFile Cesta k souboru, do něhož jsou zapsány informace o chybách a o zpracování. Parametr je nepovinný a není-li definován, je protokol zapsán do standardního souboru "System.err". Pokud se uvede hodnota "null", soubor se nevytváří a při prvním hlášení o chybě program ukončí činnost výjimkou (varování a informační hlášení se v tomto případě ignorují).

-i xmlFile Cesta k souboru s XML dokumentem, který má být zpracován.

Příklad:

```
set CLASSPATH=C:\mylib;C:\lib\syntea_xdef3.1.jar
java cz.syntea.xdef.util.XValidate –d c:\p\xdef.xml –x Davka –i c:\d\myxml.xml
```

V našem příkladě se předpokládá, že soubor "c:\d\myxml.xml" bude validován podle X-definice "Davka" ze souboru "c:\p\xdef.xml".

11 Spuštění režimu konstrukce z příkazové řádky

Režim konstrukce je možné spustit z příkazové řádky příkazem "cz.syntea.xdef.util.XCompose" ze souboru "syntea_xdef3.1.jar". Parametry při spuštění z příkazové řádky mají formát:

```
[-h] -d defList [-x xDefName] [-r rootName] [-l logFile] [-e encoding]
–i xmlFile -o outFile
```

Význam parametrů:

-h Help (vytiskne soubor s informací o parametrech spuštění). Ostatní parametry jsou v tomto případě ignorovány.

-d defList Parametrem se popíše cesta k souboru X-definic (nebo seznamu souborů, oddělených čárkou). V zápisu jména souboru je možné použít znaky „*“ a „?“, pomocí kterých je možné popsat výběr skupiny souborů. Parametr je pro režim kompozice povinný.

-x xDefName Jméno X-definice, která má být použita pro vstupní data. Parametr je nepovinný a pokud není uveden, použije se jméno kořenového elementu xml souboru.

-r rootName Jméno kořenového elementu, pokud se liší od jména, které má být použito pro vstupní data. Parametr je nepovinný a není-li uveden, dosadí se jméno převzaté z kořenového elementu vstupních dat.

-l logFile Cesta k souboru, do něhož jsou zapsány informace o chybách a o zpracování. Parametr je nepovinný a není-li definován, je protokol zapsán do standardního souboru „System.err“. Pokud se uvede hodnota „null“, soubor se nevytváří a při prvním hlášení o chybě program ukončí činnost výjimkou (varování a informační hlášení se v tomto případě ignorují).

-o outFile Soubor, do kterého budou zapsána vytvořená data.

-e encoding Jméno kódové tabulky pro výstupní soubor. Není-li kódování uvedeno, dosadí se „UTF-8“.

-i xmlFile Soubor se vstupními XML daty.

Příklad:

```
set CLASSPATH=C:\mylib;C:\lib\ syntea_xdef3.1.jar
 java cz.syntea.xdef.util.XCompose
 -o c:\temp\Priklad1_vysl.xml
 -x Seznam#Seznam_jmen
 -d c:\temp\Priklad1.xdef
 -i c:\temp\Priklad1.xml
```

Poznámka: z důvodů čitelnosti jsou parametry uvedeny na jednotlivých řádcích s úvodní mezerou, ale v příkazové řádce musí být samozřejmě na jedné řádce.

Uvedený příklad vezme data ze souboru "c:\temp\Priklad1.xml" a vytvoří výstupní soubor "c:\temp\Priklad1_vysl.xml" v kódování UTF-8 podle X-definice "Seznam" a modelu elementu <Seznam_jmen> v souboru "c:\temp\Priklad1.xdef".

12 Spuštění z Java programu

Programové prostředky pro X-definice jsou implementovány v jazyce Java a jsou dodávány v souboru:

"syntea_xdef3.1.jar",

Základní knihovna pro použití X-definic je v balíku "cz.syntea.xd".

Předpokládá se, že X-definice jsou programu předány ve zdrojovém tvaru (většinou jako soubory, které obsahují X-definice ve formátu XML). Zdrojový tvar X-definic je třeba nejprve přeložit do vnitřního tvaru, který je uložen do objektu, obsahujícího množinu X-definic ("cz.syntea.xdef.XDPool"). Je rozumné tento objekt vytvořit na začátku zpracování dat, protože překlad X-definic vyžaduje určité prostředky a čas (zejména není třeba provádět překlad definic opakovaně v cyklu). Výhodné je vytvořit na začátku programu jeden objekt "XDPool" obsahující množinu všech X-definic, které mohou být programem použity a z níž lze jednotlivé X-definice vybírat. Objekt "XDPool" lze vytvořit ze zdrojového tvaru ze souborů, jejichž jména mohou obsahovat i znaky skupinového výběru ("*" nebo "?" - např. "C:/data/*.xdef" vybere z daného adresáře všechny soubory s příponou "xdef"). Objekt "XDPool" vytvoříme pomocí statických metod třídy "cz.syntea.xdef.XDFactory". Uveďme typickou sekvenci příkazů pro tvorbu objektu "XDefPool":

```
String[] defFileNames; // the array of file pathnames
Properties props; //properties (may be null)
// Array of objects with external methods refered in X-definitions.
Class[] externals = new Class[]{this.class, com.myproject.MyClass.class};
```

```
// create the pool of X-definitions
XDPool xp = XDFactory.compileXD(props, defFileNames, externals);
```

Poznámka: Místo jména souboru může být uveden i URL odkazující na internetovou adresu X-definice. V některých případech je naopak výhodné, že konstruktor umožňuje, aby byl místo jména souboru uveden znakový řetězec, obsahující přímo zdrojový tvar X-definice. Program tuto skutečnost rozpozná podle toho, že první znak položky je "<":

```
String source = "<xd:def xmlns:xd = \"http://www.syntea.cz/xdef/3.1\">"
  + " ... "
  + "</xd:def>";
cz.syntea.xdef.XDPool xp = cz.syntea.xdef.XDFactory.compileXD(props, source);
```

Pro práci s XML dokumenty třeba vytvořit objekt "cz.syntea.xdef.XDDocument". Spuštění zpracování umožňují metody "xparse". Této metodě je třeba předat objekt s XML daty (podobně jako při překladu X-definic je zde možné zadat buď jméno souboru, URL, nebo přímo řetězec znaků s XML dokumentem místo jména souboru). Pokud je potřeba mít protokol o případných chybách při zpracování, je třeba proceduře předat také objekt s tzv. reportérem, který umožní vytvořit soubor s protokolem o chybách. Výsledkem metody je objekt "org.w3c.dom.Element": (Co/jaký element výsledek obsahuje? Odp.: Je to root element)

```
XDPool xp;
....
String xdefName;
String data;
...
try {
  XDDocument xdoc = xp.createXDDocument(xdefName);
  Element element = xdoc.xparse(data, null);
   System.out.println("OK");
} catch (Exception ex) {
   ex.printStackTrace(System.err);
}
```

Poznámka: Pro práci se zprávami je k dispozici knihovna "cz.syntea.xdef.sys". Tato knihovna obsahuje bohatou výbavu metod práce s protokolem, včetně podpory vícejazyčných aplikací. Výhodné je použít třídu ArrayReporter, která protokol ukládá do paměti. Podrobný popis je uveden v přiložené programové dokumentaci. Kromě práce se zprávami v protokolu je k dispozici množství užitečných metod, jejichž popis je v programové dokumentaci.

Pro režim konstrukce slouží metoda "xcreate" implementovaná v objektu XDDocument. Typické použití má vstupní data uložena v XML elementu. Výsledek je metodou vrácen opět jako objekt typu Element. Metodě "xcreate" je třeba předat parametr se jménem modelu elementu, podle něhož má být výsledek sestaven (jméno kořenového elementu výsledku):

```
import cz.syntea.xdef.reporter.ArrayReporter;
import cz.syntea.xdef.xml.KXmlUtils;
import cz.syntea.xdef.XDPool;
import cz.syntea.xdef.XDDocument;
import org.w3s.dom.Document;
import org.w3c.dom.Element;
...
XDPool xp;
String xdefName;
String sourceName;
String resultModelName;
...
XDDocument xdoc = xp.createXDDocument(xdefName);
Document source = KXmlUtils.parsexlm(xdefSourceFileName);
ArrayReporter reporter = new ArrayReporter();
xdoc. setXDContext(source);
Element result = xdoc.xcreate(resultModelName, reporter);
if (reporter.errors()) {
   System.err.println(reporter.printToString());
}
```

Poznámka: V uvedeném příkladě je element vytvořen ze vstupních dat pomocí parseru z knihovny "cz.syntea.xdef.xml" (uživatel však může použít i standardní prostředky např. z knihovny javax).

Pro ilustraci režimu konstrukce uveďme konkrétní X-definici, vstupní data i výpis výsledného elementu.

X-definice:

```
<?xml version="1.0" encoding="windows-1250" ?>
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1"
        xd:name="SmlouvaOutput">
```

```
<xd:declaration xd:scope="global"> <![CDATA[
/* Metoda pro kontrolu rodného čísla */
boolean rodneCislo(String s) {
   if (s.length() < 10)
      return error('Rodného čislo  je příliš krátké');
   if (s.length() > 11)
      return error('Rodného čislo  je příliš dlouhé');
   if (s.substring(6,7) != '/')
      return error('V rodném čísle chybí lomítko');
   if (!isNumeric(cut(s,5)))
      return error('Nenumerická hodnota v datové části rodného čísla');
   if (!isNumeric(s.substring(7)))
      return error('Nenumerická hodnota v indexu rodného čísla');
   return true; /* Rodné číslo je OK*/
}
]]>
</xd:declaration>

<Smlouva cps = "num(10)" >
  <Vlastnik  Nazev     = "string(1,30); create from('@nazev')"
             IC        = "num(8); create from('@ic')"
             xd:script = "create from('Klient[@role=\'1\']')"/>
  <Drzitel   Jmeno    = "string(1,30); create from('@Jmeno')"
             Prijmeni = "string(1,30); create from('@Prijmeni')"
             RC       = "rodneCislo(getText()); create from('@rc')"
             xd:script = "create from('Klient[@role=\'2\']')"/>
  <Pojistnik Nazev     = "string(1,30); create toString(from('@Jmeno')) + ' ' + from('@Prijmeni')"
             IC        = "num(8); create from('@ic')"
             xd:script = "create from('Klient[@role=\'3\']')"/>
</Smlouva>

</xd:def>
```

Vstupní data:

```
<?xml version="1.0" encoding="windows-1250" ?>
<Smlouva cps = "0123456789">
  <Klient role = "1"
          typ = "P"
          nazev = "Firma XYZ"
          ic = "12345678" />
  <Klient role = "2"
          typ = "O"
          typid = "1"
          Jmeno = "František"
          Prijmeni = "Novák"
          rc = "311270/1234" />
  <Klient role = "3"
          typ = "O"
          typid = "2"
          Jmeno = "František"
          Prijmeni = "Novák"
          rc = "311270/1234"
          ic = "87654321" />
</Smlouva>
```

Výsledek:

```
<Smlouva     cps      = "0123456789">
  <Vlastnik  Nazev    = "Firma XYZ"
             IC       = "12345678"/>
  <Drzitel   Jmeno    = "František"
             Prijmeni = "Novák"
             RC       = "311270/1234"/>
  <Pojistnik Nazev    = "František Novák"
             IC       = "87654321"/>
</Smlouva>
```

Podrobná programová dokumentace je dodávána pro jednotlivé verze prostředí Java v souborech "syntea_common_200_jdk1x_doc.zip", "syntea_xdef_200_jdk1x_doc.zip".

12.1 Ukázka spuštění v režimu validace

V našem příkladě jsou soubory, tvořící množinu X-definic, přístupné pomocí zápisu "C:\data*.xdef" a soubor se vstupními daty je "C:\data \data.xml". X-definice, která popisuje data má jméno "SouborD1A". V Java třídě "Test" jsou externí uživatelské metody, použité v X-definicích. Následuje ukázka programu pracujícího ve validačním režimu:

```
// 1. mnozina souboru s X-definicemi, které tvori množinu X-definic
File[] defs = cz.syntea.comon.sys.Sutils.getFileGroup("C:\\data\\*.xdef");

// 2. mnozina objektu s externimi uzivatelskymi metodami, které jsou pouzity
//    v X-definicich
Class[] externs = new Class[]{Test.class};

// 3. Přečteme soubor definic a vytorime množinu definic (XDPool).
Properties props = new Properties();
XDPool xpool = XDFactory(props, defs, externs);


// 4. připravíme reporter pro zápis protokolu o chybách,
FileReportWriter reporter = new  FileReportWriter ("c:/temp/errors.log");

// 5. Připravime objekt XDDocument.
XDDocument xdoc = xpool.createXDDocument(" SouborD1A");

// 6. Připravíme vstupni data
File data = new File("C:\\data\\data.xml ");

// 7. Spustíme proces validace. Výsledkem je Element
Element vysledek = xdoc.xparse(data, reporter);

// 8. Otestujeme zda procesor nahlásil chyby
if (reporter.errors()) {
    System.out.println("Chyby v souboru C:\\data\\data.xml ");
    ReportReaderInterface reader = reporter.getReportReader();
    Report rep;
    // Vytiskneme protokol chyb.
    while ((rep = reader.getReport()) != null)
        System.out.println(rep.toString());
} else {
    System.out.println("Soubor C:\\data\\data.xml je bez chyb!");
    // zpracování výsledku ...
}
```

12.2 Ukázka spuštění v režimu konstrukce

V ukázce je množina X-definic vytvořena ze souboru "c:\data\form.xdef", jméno X-definice není specifikováno (je v XDPool jediná). V X-definici nejsou použity externí uživatelské metody. V záhlaví nemusíme pro režim konstrukce uvádět <root> element (jméno požadovaného modelu se uvádí jako parametr v metodě "xcreate"). Zdrojový dokument v souboru"c:\data\zdroj.xml". Ze zdrojového souboru s údaji o zaměstnancích chceme vytvořit nový soubor, který obsahuje pouze jména zaměstnanců:

```
// 1. Připravime objekt XDDocument.
XDDocument xdoc = XDFactory(props, def, null).createXDDocument();

// 2. připravíme reporter pro zápis protokolu o chybách do paměti.
ArrayReporter reporter = new ArrayReporter();

// 3. Vytvoříme objekt s kontextem (vstupními daty) a připojíme jej k objektu XDDocument.
File data = new File("c:\data\zdroj.xml");
xdoc.setXDContext(KXmlUtils.parseXml(data));

// 4. Spustíme režim skládání a získáme výsledek
String nsUri = null; // namespace root elementu, který chceme vytvořit
String name = "root"; // jméno elementu, který chceme vytvořit
Element vysledek = xdoc.xcreate(nsUri, name, reporter);
if (reporter.errorWarnings()) {
    System.out.println(reporter.printToString());
}
```

Výpis vstupního souboru "C:\data\form.xdef":

```
<xd:def xmlns:xd = "http://www.syntea.cz/xdef/3.1">
<Seznam_jmen xd:script = "create from('Dávka')" >
```

```
  <Zaměstnanec xd:script = "occurs *"
               jméno     = "required string(1,30)"
               příjmení  = "required string(1,30)" />
</Seznam_jmen>
</xd:def>
```

Výpis souboru "C:\data\zdroj.xml":

```
<Dávka>
  <Zaměstnanec jméno    = "Jan"
               příjmení = "Novak"
               RC       = "490217/123"
               idCislo  = "1234"
               plat     = "19200">
    <Charakteristika>
        Dobrý pracovník
    </Charakteristika>
  </Zaměstnanec>

  <Zaměstnanec jméno    = "Vladimir"
               příjmení = "Valouch"
               RC       = "541121/456"
               idCislo  = "8902"
               plat     = "5300">
    <Charakteristika>
        Špatný pracovník
    </Charakteristika>
  </Zaměstnanec>

</Dávka>
```

Výsledek:

```
<Seznam_jmen>
  <Zaměstnanec jméno    = "Jan"
               příjmení = "Novak" />"
  <Zaměstnanec jméno    = "Vladimir"
               příjmení = "Valouch" />"
</Seznam_jmen>
```

12.3 Ukázka uložení a načtení souboru s X-definicemi

Vytvořený objekt XDPool můžeme uložit jako soubor a později jej použít. Binární tvar XDPool umožňuje předávat soubor s vytvořenými Definicemi v komprimované formě bez zdrojového tvaru X-definic a opětovné získání XDPool je s ohledem na čas procesoru podstatně rychlejší.

```
// 1. Vytvoření objektu XDPool
XDPool xpool = XDFactory(props, def, null);

// 2. Uložení XDPool do paměti
FileOutputStream os = new FileOutputStream("c:\data\pool.bin");
xpool.writeXDPool(os);
os.close();

// 3. Načtení XDPool ze souboru
FileInputStream is = new FileInputStream("c:\data\pool.bin");
xpool = XDFactory. readXDPool(is);
is.close();
```

12.4 Ukázka validace velkých souborů

Pokud v aplikaci pracujeme s daty, které přesahují velikost operační paměti počítače, můžeme použít zpracování v "proudovém" režimu. To umožňuje metoda "setStreamWriter" která je implementována v XDDocument. V tomto případě je výsledný XML dokument průběžně ukládán do souboru.

```
// 1. Vytvoření objektu XDDocument
XDDocument xdoc = XDFactory(props, def, externs). createXDDocument(xname);

// 2. Nastavení proudového režimu
FileOutputStream out = new FileOutputStream("c:\data\result.xml");
String encoding = "windows-1250";
xdoc. setStreamWriter(out, encoding, true);
```

13 Kontrola správnosti X-definice

Při tvorbě X-definic je vhodnou pomůckou kontrola její správnosti. Správnost X-definice je možné ověřit spuštěním programu "cz.syntea.xdef.util.CheckXdef" ze souboru "syntea_xdef3.1.jar", kterému je X-definice předána pomocí parametru. Výstupem programu je seznam chybových hlášení, nebo "listing", ve kterém jsou označena místa, kdy jsou detekovány chyby (pokud je zadán parametr "-v". Výstup je proveden do standardního souboru "System.out". Pokud jsou v testované X-definici použity externí uživatelské metody, musí být příslušné třídy dostupné v parametru "classpath" při volání programu. Pokud jsou v X-definici odkazy na další X-definice, je třeba předat cestu k nim pomocí dalších parametrů (je povolena i specifikace množiny souborů pomocí zápisu s hvězdičkou).

Parametry volání programu jsou:

```
[-h] [-v] [-d množina_X-definic] X-definice
```

Význam parametrů:

-h Pokud je uveden tento parametr, program pouze vypíše krátkou informaci o parametrech při spuštění.

-v Parametr je nepovinný. Je-li uveden, provede se výpis s podrobným výpisem o chybách (listing). Není-li uveden, provede se zkrácený výpis (seznam chyb).

X-definice Parametr je povinný a definuje se jím vstupní soubor(y) s X-definicí.

-d množina_X-definic Parametr je nepovinný a obsahuje seznam souborů s X-definicemi pro vytvoření příslušné množiny, oddělených čárkou. Nejsou-li definovány, pak množina obsahuje jedinou X-definicí nebo kolekci X-definic definovanou jedním souborem. Specifikace souboru může také obsahovat skupinové znaky („*“, „?“) a takto lze rovněž specifikovat množinu souborů s X-definicemi nebo kolekcemi X-definic.

14 Přeformátování zdrojového tvaru X-definice

Zdrojový tvar X-definice je možné automaticky přeformátovat tak, aby výsledek byl přehlednější a čitelnější. K tomu účelu slouží program "cz.syntea.xdef.util.PrettyXdef" ze souboru "syntea_xdef3.1.jar", který přečte danou X-definici a urovná vzhled modelů elementů a jejich potomků i atributů tak, aby výsledek byl přehledný a aby odsazení vnořených prvků bylo standardní. Parametry programu jsou následující:

```
[-d outDir] [-i n] [-e encoding] [-p prefix] sourceFile
```

Význam parametrů:

-h Pokud je uveden tento parametr, program pouze vypíše krátkou informaci o parametrech při spuštění.

-i n Nepovinný parametr, pomocí něhož můžeme nastavit počet mezer, použitých k odsazení vnitřních elementů, přednastavená hodnota je 3.

-e encoding Nepovinný parametr, jímž můžeme nastavit znakovou sadu kódování (hodnota musí být v slučitelná s hodnotou parametru "encoding" v záhlaví XML, např. "UTF-8", "windows-1250" a pod.). Není-li tento parametr specifikován, je použito původní kódování.

-p prefix Nepovinný parametr, pomocí něhož lze předefinovat prefix pro jmenný prostor X-definic. Není-li tento parametr specifikován, je použit původní jmenný prostor.

sourceFile Vstupní soubor nebo skupina souborů (je možné použít zápisu se znakem „*“).

Není-li uveden parametr s výstupním souborem, vstupní soubor se přepíše (Pokud během činnosti programu v tomto případě dojde k výpadku, je původní soubor přejmenován na soubor s koncovkou "bak"). Příklad:

```
set CLASSPATH=C:\mylib;C:\lib\ syntea_xdef3.1.jar
java cz.syntea.xdef.util.PrettyXdef c:\data\myxml.xml
```

Vstupní soubor "c:\data\myxml.xml" se přeformátuje a uloží na původní místo. Parametr odsazení se použije standardní, tj. 3 mezery.

15 Převod mezi XML schématy a X-definicemi

15.1 Převod z XML schémat do X-definic

Pro převod XML schémat do X-definic slouží třída XsdToXdef, pro zavolání programu pro převod jsou následující parametry:

Krátký název	Dlouhý název	Hodnota	Popis
-in	--input	CESTA	Povinný. Cesta k souboru XML schématu, které se má převést.
-out	--output	CESTA	Povinný. V případě, že je nastavena vlastnost pro převod schémat do jednotlivých souborů X-definic, uvádí cestu ke složce, kam se výsledné soubory uloží. V jiném případě uvádí cestu a název výstupného souboru s kolekcí X-definic.
-s	--separated	není	Volitelný. Oznamuje programu, že má převést schémata do jednotlivých souborů X-definic. Standardně je tato vlastnost vypnuta.
-d	--debugMode	není	Volitelný. Zapíná ladící mód, při kterém program vypisuje aktuálně zpracovávané elementy schémat a X-definic. Standardně je tento mód vypnut.
-p	--xdefPrefix	PREFIX	Volitelný. Nastavuje prefix pro uzly X-definic. Výchozí hodnota je "xd".
-xd	--xdefVersion	VERZE	Volitelný. Nastavuje verzi výstupního dokumentu X-definic. Výchozí verze je 2.0 (zatím jediná podporována).
-?, -h	--help	není	Volitelný. Vytiskne nápovědu pro použití příkazu.

Parametry převodu:

- Prefix pro elementy X-definic (výchozí hodnota je "xd").
- Verze X-definic na výstupu (výchozí je 2.0).
- Zapnutí nebo vypnutí ladicího režimu.

Existují 2 módy převodu. První mód vytváří kolekci X-definic, která obsahuje množinu X-definic. Výsledný soubor s kolekcí X-definic se uloží pod názvem, předaným jako parametr. Druhý mód převádí každé schéma do samostatného souboru X-definic. Množina všech vytvořených souboru se uloží do složky s názvem, který se předá dané metodě jako parametr.

15.2 Převod z X-definic do XML schémat

Pro převod X-definic do XML schémat slouží třída XdefToXsd, pro zavolání programu pro převod jsou následující parametry:

Switch	Hodnota	Popis
-i	CESTA	Seznam cest k souborům X-definic.
-o	CESTA	Povinný. Adresář, kam se budou generovat XML schémata.
-m	jméno	Volitelný. Jméno root modelu
-x	jméno	Volitelný. Jméno X-definice, ze které se začnou generovat XML schémata.
-? –h		Vyvolání nápovědy.

Množina všech vytvořených souborů se uloží do složky s názvem, který se předá dané metodě jako parametr -o.

Příloha A: X-definice X-definic

Jazyk X-definic je možné formálně popsat pomocí X-definice. V následující X-definici jsou pro popis textových hodnot, obsahujících skript, použity metody zpracovávající syntaktická pravidla, popsaná gramatikou skriptu. Předpokládá se, že nejdříve jsou všechny textové hodnoty zpracovány makroprocesorem, který případné odkazy na makra nahradí

jejich hodnotou. Odkaz na makro začíná posloupností písmen "${", za níž následuje jméno odkazovaného makra případně následované specifikací parametrů v závorkách a celý odkaz je ukončen znakem "}". Zápis makra a na odkazu makro je popsán v odstavci 3.2.27. X-definice X-definic je v následujícím XML dokumentu:

```
<!--  For the X-definition of X-definition is used the metanamespace: "METAXDef" -->

<xd:def xmlns:xd   = "METAXDef"
  xmlns:XD         = "http://www.syntea.cz/xdef/3.1"
  root             = "XD:def | XD:collection"
  XD:metaNamespace = "METAXDef" >

<xd:declaration>
  /* Default namespace of X-definition */
  final String xdUri = "http://www.syntea.cz/xdef/3.1";
  /* Variable id used as the namespace of checked source; default is xdUri.*/
  String NS = xdUri;

/*****************************************************************************
* Types of values see BNF grammar below
*****************************************************************************/
  type classList XDScript.rule('ClassList');
  type methodList XDScript.rule('MethodList');
  type rootList XDScript.rule('RootList');
  type xdefScript XDScript.rule('XdefScript');
  type declarationScript XDScript.rule('DeclarationScript');
  type valueScript XDScript.rule('ValueScript');
  type attributeScript XDScript.rule('AttributeScript');
  type elementScript XDScript.rule('ElementScript');
  type groupScript XDScript.rule('ElementScript');
  type groupModelScript XDScript.rule('ElementScript');
  type Occurrence XDScript.rule('Occurrence');
  type elementCreateExpression XDScript.rule('ElementCreateSection');
  type xdIdentifier XDScript.rule('Identifier');
  type refName XDScript.rule('RefName');
  type booleanLiteral XDScript.rule('BooleanLiteral');

  /** check element name and namespace URI (used in match section) */
  boolean xdName(String name) {
    return getElementLocalName() EQ name AND getNamespaceURI() EQ NS;
  }
</xd:declaration>

  <XD:collection xd:script = "init NS = @metaNamespace ?
                              (String) @metaNamespace : xdUri;
                              options moreAttributes"
    metaNamespace = "optional uri; options acceptQualifiedAttr" >
  <!--
     Names of other atributes (option moreAttributes) must start with "impl-"
   -->

    <XD:def xd:script = "occurs +; ref XD:def" />

  </XD:collection>

  <XD:def xd:script = "init NS = @metaNamespace ? (String) @metaNamespace : xdUri;
                       options moreAttributes"
    name            = "optional QName; options acceptQualifiedAttr"
    metaNamespace   = "optional uri; options acceptQualifiedAttr"
    include         = "optional uriList; options acceptQualifiedAttr"
    root            = "optional rootList; options acceptQualifiedAttr"
    script          = "optional xdefScript; options acceptQualifiedAttr" >
  <!-- Names of other atributes (option moreAttributes) must start with "impl-" -->

    <xd:mixed>

      <XD:macro xd:script = "occurs *; options moreAttributes"
        name  = "required QName; options acceptQualifiedAttr"
        xd:attr = "occurs * xdIdentifier" >
        optional string;
      </XD:macro>

      <XD:BNFGrammar xd:script = "occurs *"
         extends = "optional xdIdentifier; options acceptQualifiedAttr"
         name = "xdIdentifier; options acceptQualifiedAttr">
         required @extends || checkBNF; /* don't check extended grammar.*/
```

```
    </XD:BNFGrammar>

    <XD:declaration xd:scope="global" xd:script = "occurs *">
       optional declarationScript;
    </XD:declaration>

    <xd:choice occurs = "*">
        <XD:choice xd:script = "occurs *;
                                match @name || @XD:name; ref XD:choiceDef"
             name = "required QName; options acceptQualifiedAttr" />
        <XD:mixed xd:script="occurs *;
                            match @name || @XD:name; ref XD:mixedDef"
          name = "required QName; options acceptQualifiedAttr" />
        <XD:sequence xd:script="occurs *;
                                match @name || @XD:name; ref XD:sequenceDef"
          name = "required QName; options acceptQualifiedAttr" />
        <XD:list xd:script="occurs *; match @name || @XD:name; ref XD:listDef"
          name = "required QName; options acceptQualifiedAttr" />
        <XD:any xd:script="occurs *; match @name; ref XD:anyDef" />

        <xd:any xd:script = "occurs *; ref xelement" />
          optional valueScript;
    </xd:choice>
  </xd:mixed>

</XD:def>

<!-- model elementu -->
<xelement xd:script = "match getNamespaceURI() NE NS; options moreAttributes"
  xd:attr   = "occurs * attributeScript"
  xd:text   = "occurs * valueScript"
  XD:script = "optional elementScript" >
  <xd:choice occurs = "*" ref = "xcontent" />
</xelement>

<xd:choice name = "xcontent">
  <XD:choice   xd:script = "occurs *; match @ref || @XD:ref; ref XD:choiceRef" />
  <XD:choice   xd:script = "occurs *; ref XD:choiceDef" />
  <XD:mixed    xd:script = "occurs *; match @ref || @XD:ref; ref XD:mixedRef" />
  <XD:mixed    xd:script = "occurs *; ref XD:mixedDef" />
  <XD:sequence xd:script = "occurs *; match @ref || @XD:ref; ref XD:sequenceRef" />
  <XD:sequence xd:script = "occurs *; ref XD:sequenceDef" />
  <XD:list     xd:script = "occurs *; match @ref || @XD:ref; ref XD:listRef"/>
  <XD:any      xd:script = "occurs *; match @XD:ref; ref XD:anyRef"/>
  <XD:any      xd:script = "occurs *; match !@XD:ref; ref xelement"/>
  <xd:any      xd:script =  "occurs *; ref xelement" />
  <XD:text> optional valueScript; </XD:text>
  optional valueScript;
</xd:choice>

<XD:choiceRef occurs = "optional Occurrence;
            options acceptQualifiedAttr"
  script = "optional string; options acceptQualifiedAttr"
  create = "optional elementCreateExpression; options acceptQualifiedAttr"
  ref    = "required QName; options acceptQualifiedAttr" />

<XD:choiceDef occurs = "optional Occurrence; options acceptQualifiedAttr"
  script = "optional groupScript; options acceptQualifiedAttr"
  create = "optional elementCreateExpression;
            options acceptQualifiedAttr"
  ref    = "illegal; options acceptQualifiedAttr" >
   <xd:choice ref = "xcontent" occurs = "*" />
</XD:choiceDef>

<XD:sequenceRef occurs = "optional Occurrence; options acceptQualifiedAttr"
  create = "optional elementCreateExpression; options acceptQualifiedAttr"
  script = "optional groupScript; options acceptQualifiedAttr"
  ref    = "required QName; options acceptQualifiedAttr" />

<XD:sequenceDef occurs = "optional Occurrence;
            options acceptQualifiedAttr"
  create = "optional elementCreateExpression; options acceptQualifiedAttr"
  script = "optional groupScript; options acceptQualifiedAttr"
  ref    = "illegal; options acceptQualifiedAttr" >
  <xd:choice ref = "xcontent" occurs = "*" />
</XD:sequenceDef>
```

```
  <XD:mixedRef ref = "required QName; options acceptQualifiedAttr"
    empty  = "optional booleanLiteral; options acceptQualifiedAttr"
    script = "optional groupScript; options acceptQualifiedAttr"
    create = "optional elementCreateExpression; options acceptQualifiedAttr" />

  <XD:mixedDef ref = "illegal; options acceptQualifiedAttr"
    script = "optional groupScript; options acceptQualifiedAttr"
    empty  = "optional booleanLiteral; options acceptQualifiedAttr"
    create = "optional elementCreateExpression; options acceptQualifiedAttr" >
    <xd:choice xd:script = "+; ref xcontent;" />
  </XD:mixedDef>

  <XD:listRef ref = "required refName; options acceptQualifiedAttr" />

  <XD:listDef ref = "illegal refName; options acceptQualifiedAttr">
    <xd:choice xd:script = "*; ref xcontent;" />
  </XD:listDef>

  <XD:anyDef name= "required QName"
    script = "optional groupScript; options acceptQualifiedAttr" />
  <XD:anyRef script = "optional groupScript" />

/*****************************************************************************
* Declaration of BNF grammar of X-definition script
/*****************************************************************************/
  <xd:BNFGrammar name = "XDScript">
<![CDATA[
Letter ::= $letter

Char ::= $xmlChar

WhiteSpace ::= $whitespace

Comment ::= "/*" ( [^*]+ | "*" [^/] )* "*/"

S ::= ( WhiteSpace | Comment )+

/**** Keywords ****/

Keyword ::=
  "if" | "else" | "do" | "while" | "continue" | "break" | "switch" | "case" |
  "for" | "return" | "def" | "try" | "catch" | "throw" | "finally" |
  "external"|"new" | "fixed" | "required" | "implied" | "optional" | "ignore" |
  "illegal" | "occurs" | "onTrue" | "onError" | "onAbsence" | "default" |
  "onExcess" | "onStartElement" | "onIllegalAttr" | "onIllegalText" |
  "onIllegalElement" | "onIllegalRoot" | "create" | "init" | "options" | "ref" |
  "match" | "final" | "forget" | "template" | "type" | "uniqueSet" |
  "EQ" | "NE" | "LT" | "LE" | "GT" | "GE" |
  "LSH" | "RSH" | "RRSH" | "AND" | "OR" | "XOR" | "MOD" | "NOT" | "NEG" |
  "OOR" | "AAND" |
  "true" | "false" |
  "$PI" | "$E" | "$MAXINT" | "$MININT" | "$MINFLOAT" | "$MAXFLOAT" |
  "$NEGATIVEINFINITY" | "$POSITIVEINFINITY"

/**** Base symbols ****/

Digit ::= [0-9]

Identifier ::= (BaseIdentifier ( ':' BaseIdentifier)? ) - Keyword

BaseIdentifier ::= (( Letter | "_" | "$")  ( Letter | Digit | "_" )*)

RawIdentifier ::= ( Letter | "_")  ( Letter | Digit | "_" )*

QualifiedIdentifier ::= Identifier ( "." RawIdentifier )+ | Identifier

BooleanLiteral ::= "true" | "false"

DecimalInteger ::= ( Digit ("_")* )+

HexaDigit ::= Digit | [a-fA-F]

IntegerLiteral ::= DecimalInteger | ( "0x" | "0X" ) (HexaDigit ("_")* )+
/* Inside of a number specification it is possible to insert the character "_".
  This character does not influence value of number, it just makes a number more
```

```
  readable. E.g. number 123456789 you can write as 123_456_789 (or 0x0f123456
  as 0x0f_12_34_56). */

SignedIntegerLiteral ::= ("+" | "-")? IntegerLiteral

NumberLiteral ::=
  DecimalInteger ( "." DecimalInteger )? ( ("E" | "e") [-+]? DecimalInteger )?

SignedNumberLiteral ::= ( "+" | "-" )? NumberLiteral

SpecChar ::=
  "\" ("\" | [0-7]+ | '"' | "'" | "n" | "r" | "t" ) | UnicodeCharSpecification
/* The octal specification is in interval 0 .. 377*/

UnicodeCharSpecification ::= "\u" HexaDigit HexaDigit HexaDigit HexaDigit

StringLiteral ::= "'" ( "''" | [^'\] | SpecChar)* "'" |
  '"' ( '""' | [^"\] | SpecChar)* '"'
  /* Opening and closing delimiter must be either """ or "'". Occurrence of this
     delimiter inside of literal can be recorded as double delimiter or
     in the form of SpecChar. */

Literal ::= BooleanLiteral | NumberLiteral | StringLiteral

XMLName ::= $xmlName /* XMLName see XML specification */

KeyName ::= "%" XMLName

AttributeName ::= "@" XMLName

Reference ::= "ref" S RefName

RefName ::= XMLName ("#" XMLName)?

RootList ::= S? (RefName | "*") (S? "|" S? (RefName | "*"))* S?

ClassList ::= (QualifiedIdentifier (S? "," S? QualifiedIdentifier)*)? S?

MethodList ::= (MethodListItem (S? ";" S? MethodListItem?)*)? S?

MethodListItem ::= ("Java:" S?)? Identifier S?
  QualifiedIdentifier S? "(" S? MethodListItemParamList? S? ")" S?
  ("as" S? Identifier)?

MethodListItemParamList ::= QualifiedIdentifier (S?"[" S? "]")?
  (S? "," S? QualifiedIdentifier (S?"[" S? "]")?)*

MacroReference ::= "${" S? RefName S? MacroParams? S? "}"

MacroParams ::= "(" S? Identifier ( S? "," S? Identifier)* S? ")"

/**** Script language ****/

Expression ::= Expr1 (S? "?" S? Expression S? ":" S? Expression )?

OperatorLevel_1 ::= "OR" | "OOR" | "XOR" | "||" | "|" | "^"

Expr1 ::= Expr2 (S? OperatorLevel_1 S? Expr2 )*

OperatorLevel_2 ::= "AND" | "AAND" | "&&" | "&"

Expr2 ::= Expr3 (S?  OperatorLevel_2 S? Expr3 )*

OperatorLevel_3 ::=
  "LT" | "<" | "GT" | ">" | "==" | "EQ" | "LE" | "<=" | "GE" |
  ">=" | "!=" | "NE" | "<<" | "LSH" | ">>" | "RSH" | ">>>" | "RRSH"

Expr3 ::= Expr4 (S? OperatorLevel_3 S? Expr4 )*

OperatorLevel_4 ::= "*" | "/" | "%"

Expr4 ::= Expr5 (S? OperatorLevel_4 S? Expr5 )*

OperatorLevel_5 ::= "+" | "-"

Expr5 ::= Expr (S? OperatorLevel_5 S? Expr )*
```

```
Expr ::= (UnaryOperator S? | CastRequest S?)*
  (Value | Literal | "(" S? Expression S? ")") (S? "." S? Method)?

ConstantExpression ::= Literal
  /* ConstantExpression must be a Literal. */

CastRequest ::= S? "(" S? TypeIdentifier S? ")" S?

UnaryOperator ::= "+" | "-" | "!" | "NOT" | "~"

TypeIdentifier ::= "int" | "String" | "float" | "boolean" | "Datetime" |
  "Decimal" | "Duration" | "Exception" | "Context" | "Element" | "Message" |
  "Bytes" | "XmlOutStream" | "BNFGrammar" | "BNFRule" | "Parser" |
  "ParseResult" | "AnyValue"

Value ::= Constructor | "null" | Method | Increment | VariableReference |
  KeyParameterReference | Literal | AttributeName |
  "$MAXINT" | "$MININT" |
  "$MINFLOAT" | "$MAXFLOAT" |"$NEGATIVEINFINITY" | "$POSITIVEINFINITY" |
  "$PI" | "$E"

Constructor ::= "new" S TypeIdentifier S? "(" S? ParameterList? S? ")" |
  NamedValue | ContextValue

NamedValue ::= KeyName S? "=" S? Expression

ContextValueStart ::= (NamedValue (S? "," S? NamedValue)*) | Expression

ContextValue ::= "[" S? (ContextValueStart (S? "," S? Expression)*)? S? "]"

KeyParameterReference ::= KeyName

Method ::= QualifiedIdentifier S? ("(" S? ParameterList? S? ")")?

Increment ::=
  ("++" | "--") S? VariableReference | VariableReference S? ("++" | "--")
/* The variable must be declared as an integer or a float */

/* $ChkElement is old version only.
ParameterList ::=("$Element" | $Data" | Expression) (S? "," S? Expression)*
*/
ParameterList ::= (Expression) (S? "," S? Expression)*

VariableReference ::= Identifier - TypeIdentifier

MethodDeclaration ::=
  ( "void" | TypeIdentifier ) S MethodName S? ParameterListDeclaration S? Block

MethodName ::= QualifiedIdentifier

ParameterListDeclaration ::=
  "(" S? ( SeqParameter (S? "," S? SeqParameter )* )? ")"

SeqParameter ::= TypeIdentifier S ParameterName

KeyParameter ::= KeyName S? "=" S? (TypeIdentifier | ConstantExpression)

ParameterName ::= Identifier

ParentalExpression ::= S? "(" S? Expression S? ")" S?

StatementExpression ::= Expression

Statement ::= S? (Statement1 | Statement2)

Statement1 ::= Block | SwitchStatement | TryStatement

Statement2 ::= ( IfStatement | ForStatement | WhileStatement | DoStatement |
  ReturnStatement | ThrowStatement | BreakStatement | ContinueStatement |
  Method | Increment | AssignmentStatement S? ";" ) | EmptyStatement

EmptyStatement ::= ";"

StatementSequence ::= ( S? VariableDeclaration S? ";" | Statement)*
```

```
Block ::= "{" StatementSequence S? "}"

SimpleStatement ::= S? (Statement1 | (Statement2? S? ";" S?)) | EmptyStatement

IfStatement ::=
  "if" ParentalExpression SimpleStatement ( S? "else" S? SimpleStatement)?
  /* Result of ParentalExpression must be boolean. */

ForStatement ::= "for" S? "(" S?  ForInit? S? ";" S?
  Expression? S? ";" S? ForStep? S? ")" S? SimpleStatement

ForInit ::= AssignmentStatement | VariableDeclaration

ForStepStatement ::= Method | Increment | AssignmentStatement

ForStep ::= ForStepStatement (S? "," S? ForStepStatement)*

WhileStatement ::= "while" ParentalExpression SimpleStatement
  /* Result of ParentalExpression must be boolean. */

DoStatement ::= "do" S? Statement S? "while" ParentalExpression
  /* Result of ParentalExpression must be boolean. */

SwitchStatement ::=
  "switch" ParentalExpression "{" SwitchBlockStatementVariant* S? "}"
  /* Result of ParentalExpression must be integer. Any variant may occur in the
     switch statement only once. */

SwitchBlockStatementVariant ::=
   S? ("default" | "case" S? ConstantExpression) S? ":" S? StatementSequence?
  /* Type  of ConstantExpression must be integer. */

ThrowStatement ::= "throw" S? ExceptionValue

ExceptionValue ::=
  "new" S? "Exception" S? "(" S? (Expression S?)? ")" | Identifier

TryStatement ::= "try" S? "{" S? StatementSequence S? "}" S?
  "catch" S? "(" S? "Exception" S?  Identifier S? ")" S?
  "{" S? StatementSequence S? "}"

ReturnStatement ::= "return" (S? Expression)?

BreakStatement ::= "break" (S? Identifier)?

ContinueStatement ::= "continue" (S? Identifier)?

AssignmentStatement ::= Identifier S? ((AssignmentOperator S? Expression) |
 (("=" S? Identifier))+ S? (AssignmentOperator S? Expression)) |
  S? AssignmentOperator S? Expression

AssignmentOperator ::=
  ("|" | "^" | "+" | "-" | "*" | "/" | "&" | "%") "=" | "="

VariableDeclaration ::= ("final" S | "external" S )*
  TypeIdentifier S VariableDeclarator (S? "," VariableDeclarator)*

VariableDeclarator ::= S? (AssignmentStatement | Identifier)

Occurrence ::= ("occurs" S)? ("*" | "+" | "?" |
  "required" | "optional" | "ignore" | "illegal" |
  IntegerLiteral (S? ".." (S? ( "*" | IntegerLiteral))? )?)
  /* Value of the second IntegerLiteral (after "..")  must be greater or equal
     to the first one. */

ExplicitCode ::= Block
  /* If result value is required it must be returned by command
     "return value". */

/**** Script of the header of X-definition ****/

XdefScript ::=
  S? ( XdefInitSection | XdefOnIllegalRoot | XdefOnXmlError | XdefOptions )* S?
  /* Each item can be specified only once. */

XdefInitSection ::= "init" S Statement
```

```
XdefOnIllegalRoot ::= "onIllegalRoot" S Statement

XdefOnXmlError ::= "onXmlError" S Statement

XdefOptions ::= "options" S XdefOptionsList

XdefOptionsList ::= XdefOptionItem ( S? "," S? XdefOptionItem )*
  /* Each item can be specified only once. */

XdefOptionItem ::=
  "moreAttributes" | "moreElements" | "moreText" |
  "forget" | "notForget" | "clearAdoptedForgets" |
  "resolveEntities" | "ignoreEntities" |
  "resolveIncludes" | "ignoreIncludes" |
  "preserveComments" | "ignoreComments" |
  "acceptEmptyAttributes" |
  "preserveEmptyAttributes" | "ignoreEmptyAttributes" |
  "preserveAttrWhiteSpaces" | "ignoreAttrWhiteSpaces" |
  "preserveTextWhiteSpaces" | "ignoreTextWhiteSpaces" |
  "setAttrUpperCase" | "setAttrLowerCase" |
  "setTextUpperCase" | "setTextLowerCase" |
  "acceptQualifiedAttr" | "notAcceptQualifiedAttr" |
  "trimAttr" | "noTrimAttr" |
  "trimText" | "noTrimText" |
  "resolveEntities" | "ignoreEntities" |
  "resolveIncludes" | "ignoreIncludes" |
  "preserveComments" | "ignoreComments"

/**** Script of text nodes and attributes ****/

AttributeScript ::= ValueScript

ValueScript ::= ValueExecutivePart*

ValueExecutivePart::= S? ( ValueValidationSection | ValueInitSection |
  ValueOnTrueSection | ValueOnErrorSection | ValueOnAbsenceSection |
  ValueDefaultSection | ValueCreateSection | ValueFinallySection |
  AttributeOptions | Reference | S | ";" ) S?
  /* Each item can be specified only once. */

ValueValidationSection ::=
 (Occurrence S? CheckValueSpecification?) | CheckValueSpecification

CheckValueSpecification ::= ExplicitCode | Expression | TypeMethodName
  /* Expression or ExplicitCode must return a value of boolean type. */

TypeMethodName ::= Identifier

ValueInitSection ::= "init" S? ( ExplicitCode | Statement )
  /* If Method or ExplicitCode returns a value, it will be ignored. */

ValueOnTrueSection ::= "onTrue" S? ( ExplicitCode | Statement )
  /* If Method or ExplicitCode returns a value, it will be ignored. */

ValueOnErrorSection ::= "onError" S? ( ExplicitCode | Statement )
  /* If Method or ExplicitCode returns a value, it will be ignored. */

ValueOnAbsenceSection ::= "onAbsence" S ( ExplicitCode | Statement )
  /* If Method or ExplicitCode returns a value, it will be ignored. */

ValueCreateSection ::= "create" S? ( ExplicitCode | Expression )
  /* Expression nebo ExplicitCode must return value of String. */

ValueDefaultSection ::= "default" S? ( ExplicitCode | Expression )
  /* Expression nebo ExplicitCode must return value of String. */

ValueFinallySection ::= "finally" S? ( ExplicitCode | Statement )
  /* If Method or ExplicitCode returns a value, it will be ignored. */

AttributeOptions ::= ValueOptions

ValueOptions ::= "options" S? ValueOptionsList

ValueOptionsList ::=  ValueOption ( S? "," S? ValueOption )*
  /* Each item can be specified only once. */
```

```
ValueOption ::= "preserveTextWhiteSpaces" | "ignoreTextWhiteSpaces" |
  "setTextUpperCase" | "setTextLowerCase" | "trimText" | "noTrimText" |
  "preserveAttrWhiteSpaces" | "ignoreAttrWhiteSpaces" |
  "setAttrUpperCase" | "setAttrLowerCase" | "trimAttr" | "noTrimAttr"

/**** Script of elements ****/

ElementScript ::= ElementExecutivePart*

ElementExecutivePart ::= S? ( Occurrence | ElementVarSection |
  ElementMatchSection| ElementInitSection | ElementOnAbsenceSection |
  ElementOnExcessSection | ElementCreateSection | ElementFinallySection |
  ElementOptions | Reference | ElementForgetSection | ElementOnStartSection |
  S | ";" )
  /* Each item can be specified only once.
   * If Occurrence is not specified, the implicite value is "required" */

ElementVarSection ::= "var" S?
  (("{"(S? VariableDeclaration S? ";" S?)* "}") | VariableDeclaration S? ";" S?)

ElementMatchSection ::= "match" S? ( Expression | ExplicitCode )
  /* Expression nebo ExplicitCode must return value of boolean. */

ElementOnStartSection ::= "onStartElement" S? Statement?

ElementOnExcessSection ::= "onExcess" S? Statement?

ElementOnAbsenceSection ::= "onAbsence" S? Statement?

ElementCreateSection ::= "create" S? (Expression | ExplicitCode)
  /* Expression nebo ExplicitCode must return value of Context or Element. */

ElementFinallySection ::= "finally" S? Statement?

ElementInitSection ::= "init" S? Statement?

ElementForgetSection ::= "forget"

ElementOptions ::= "options" S? ElementOptionsList

ElementOptionsList ::=  ElementOptionItem ( S?  "," S? ElementOptionItem )*
  /* Each item can be specified only once. */

ElementOptionItem ::= "moreAttributes" | "moreElements" | "moreText" |
  "forget" | "notForget" | "acceptEmptyAttributes" | "clearAdoptedForgets" |
  "preserveEmptyAttributes" | "ignoreEmptyAttributes" |
  "preserveAttrWhiteSpaces" | "ignoreAttrWhiteSpaces" |
  "preserveTextWhiteSpaces" | "ignoreTextWhiteSpaces" | "setAttrUpperCase" |
  "setAttrLowerCase" | "setTextUpperCase" | "setTextLowerCase" |
  "acceptQualifiedAttr" | "notAcceptQualifiedAttr" | "trimAttr" | "noTrimAttr" |
  "trimText" | "noTrimText" | "resolveEntities" | "ignoreEntities" |
  "resolveIncludes" | "ignoreIncludes" | "preserveComments" | "ignoreComments" |
  "nillable" | "noNillable"

/**** Script of declaration part ****/

DeclarationScript ::= (S? (TypeDeclaration | VariableDeclaration S? ";" |
      MethodDeclaration | ";"))* S?

TypeDeclaration ::= ("type" S Identifier S?
  ( (Identifier (S? "," S? Identifier)* S? ";" S?) | TypeDeclarationBody ))
  | ("uniqueSet" S Identifier S?  UniqusetDeclarationBody )

TypeDeclarationBody ::= ("{" S? "parse" S? ":" S?
  (ExplicitCode | (Expression S? ";"? S?)) S? "}") | Expression
  /* Expression or ExplicitCode must return value of boolean or ParseResult.*/

UniqusetDeclarationBody ::= ("{" S? Identifier S? ":" S? "?"? S? Expression
  (S? ";" S? S? Identifier S? ":" S? "?"? S? Expression)? S? ";"? S? "}")
  | Expression
  /* Expression or ExplicitCode must return value of boolean or ParseResult.*/
]]>
  </xd:BNFGrammar>

</xd:def>
```

Příloha B: Příklady

Externí metody použité v některých projektech pro ČKP a SKP

`tab(t,s)`	obor hodnot je určen v databázové tabulce "t" sloupcem "s"
`dateCurr(c)`	datumová hodnota tvaru "c" a musí být menší než běžné datum
`dateDavka(c)`	datumová hodnota tvaru "c" a musí být menší než datum dávky, resp. předání
`rokVyroby()`	datum ve tvaru "rrrr" a musí platit, že 1950<hodnota<=běžný rok
`mesicSTK()`	hodnota musí být ve tvaru "rrrr/mm"
`RC()`	hodnota musí mít tvar rodného čísla ve tvaru "rrxmdd/nnnn" nebo "rrxmdd/nnn" nebo "rrxmddnnnn" nebo "rrxmddnnn" a musí vyhovovat jeho pravidlům
`IC()`	hodnota musí mít tvar identifikačního čísla a musí odpovídat jeho pravidlům

Externí metody použité v některých projektech ČKP a SKP

X-definice řídicího souboru dávky, používaného v projektech pro ČKP, resp. SKP:

```
<?xml version="1.0" encoding="windows-1250"?>
<xd:def  xmlns:xd="http://www.syntea.cz/xdef"
         impl-version="2013_11.0" impl-date="12.11.2013"
         xd:name     ="Davka"
         xd:root     ="Davka">
 <Davka
         Odesilatel          ="required num(1,9999999)"
         Prijemce            ="required num(1,9999999)"
         Kanal               ="required string(2,5)"
         SeqDavky            ="required int()"
         SeqDavkyRef         ="optional int()"
         Vytvorena           ="required datetime('yyyyMMddHHmmss')"
         Mode                ="required list('STD','TST','DEV')"
         OperationForm       ="optional list('TRY','STD','DIR')"
         UserField           ="optional string()">
     <UserAttributes       xd:script="occurs 0..1"
            TransID          ="optional string()"
     />
     <Soubor               xd:occurs="1.."
            NazevSouboru     ="required string(1,50)"
            FormatSouboru    ="required list('PIPE','XML')"
            DruhSouboru      ="required regex('[A-Z]\\d[A-Z]{1,2}')"
            PocetZaznamu     ="required int(0,99_999_999)">
        <CheckSum
               Type          ="required list('MD5')"
               Value         ="required string(32)"
        />
     </Soubor>
 </Davka>
</xd:def>
```

Příloha C: Slovník nejdůležitějších pojmů

Model elementu Modelem elementu popis elementu s údaji o charakteru datových hodnot jednotlivých částí. Model elementu je přímý potomek X-definice.

Neuspořádaná skupina (mixed) Neuspořádaná skupina popisuje seznam, kde pořadí popsaných prvků může být libovolné

Odkaz- Odkazem rozumíme odkaz na model elementu ve tvaru jméno X-definice#jméno modelu

Skupina Skupinu tvoří zápis posloupnosti elementů resp. textových uzlů, včetně příslušného skriptu.

Textová hodnota Textovou hodnotou rozumíme hodnotu textového uzlu elementu

Typ atributu Typem atributu rozumíme textovou hodnotu atributu, která vyhovuje určitým pravidlům, popsaným ve skriptu.

Uspořádaná skupina (sequence) V uspořádaných skupinách je pořadí prvků shodné s jejich popisem. Seznam elementů modelu je standardně uspořádanou sekvencí.

Výběrová skupina (choice) Výběrová skupina umožňuje popsat různé varianty.

Příloha D: Odkazy na literaturu

[1] Jiří Kosek: XML pro každého – podrobný průvodce (Grada)

[2] Extensible Markup Language (XML) 1.0, W3C Recommendation, http://www.w3c.org/TR/REC-xml

[3] W3C Date and Time Formats, http://www.w3c.org/TR/NOTE-datetime

[4] W3C XML Schema, http://www.w3c.org/TR/xmlschema-1 , http://www.w3c.org/TR/xmlschema-2

[5] Mark Grand: Java 1.1 Referenční příručka jazyka (Grada)

[6] James Gosling, Bill Joy, Guy Steele, Gilad Bracha: The Java Language Specification, http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html