



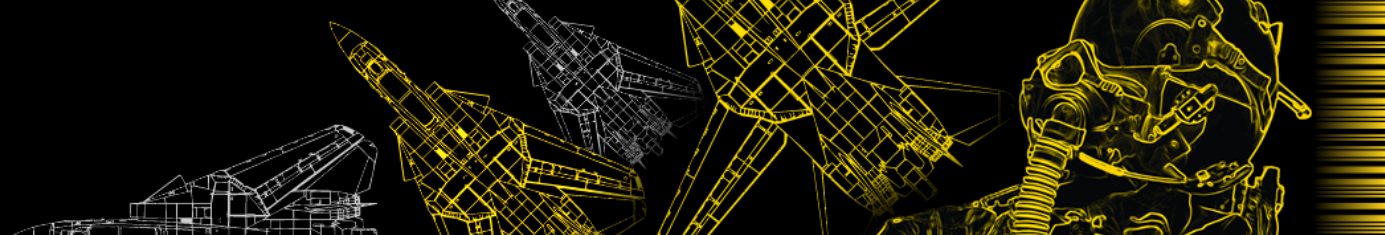
Úvod do práce s XML



10. srpna 2018
Pavel Bory

Agenda

- XML
- Kontinuální přístup
- Document Object Model (DOM)
-



XML



XML

› Historie

- › 1969 – GML – Generalized Markup Language
- › 1968 – SGML – Standard Generalized Markup Language
 - › Jazyk byl příliš komplexní, což bránilo rozšíření
 - › ISO standard
 - › Je od něj odvozeno HTML (1991)
 - › HTML 5.0 (2014)
- › 1998 – XML 1.0 (W3C)
 - › eXtensible Markup Language
 - › Odvozeno od SGML
 - › 2006 – XML 1.1
 - › Liší se restrikcemi na znaky použité v názvech elementů a atributů

› Vlastnosti

- › Odstínění logické struktury od prezentační
- › Snadná konverze
- › Validace dokumentu
- › XPath a další technologie

XML

› Použití

- › Zejména pro výměnu dat mezi aplikacemi
- › Konfigurační soubory
- › ...

XML dokument

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- komentář -->
<?xml:stylesheet type="text/css" href="style.css" ?>
<odds>
  <timestamp>1414952400023</timestamp>
  <filter>
    <![CDATA[ if( opportunity[0].rate < 1.9 && opportunity[1].rate > 4.1 ){return false;}]]>
  </filter>
  <sport name="Tenis">
    <competition name="Traralgon">
      <match id="2164326" name="Kokkinakis & G.Jones" dateClosed="03.11.2014 04:15">
        <opportunity name="1" rate="1.16"/>
        <opportunity name="2" rate="4.30"/>
      </match>
      <match id="2164329" name="J.Ward & B.Klein" dateClosed="04.11.2014 01:00">
        <opportunity name="2" rate="3.36"/>
        <opportunity name="1" rate="1.25"/>
      </match>
    </competition>
  </sport>
</odds>
```

XML dokument

› Elementy

› Uzavřené ve značkách

- › `<timestamp>text</timestamp>`
- › Jméno značka je case-sensitive

› Musí mít uzavírací značku

- › `<match><opportunity></opportunity></match>`
- › `<match><opportunity></match>`

› Prázdná značka

- › `<match><opportunity /></match>`

› Značky musí mít správné zanoření – nesmí se „křížit“

- › `<competition><match></match></competition>`
- › `<competition><match></competition></match>`

› Kořenový element dokumentu je právě jeden

› Elementy mohou obsahovat

- › 0 .. n vnořených elementů (potomků)
- › text
- › 0 .. n atributů

› XML dokument uchovává bílé znaky

XML dokument

› Jmenné konvence

› Názvy elementů

- › Mohou obsahovat písmena, čísla a další znaky
- › Nemohou začínat číslicí nebo interpunkčním znaménkem
- › Nemohou začínat sekvencí „xml“
- › Nemohou obsahovat mezery
- › Vyhněte se použití
 - › `<book-title>`
 - › `<book.title>`
 - › `<book:title>`
- › Vyhněte se použití

› Atributy

- › Doplnující informace o elementu
- › Neměly by obsahovat rozsáhlá data
- › Musí být uzavřeny v uvozovkách

```
<match id="2164326" name="kokkinakis & G.Jones" dateClosed="03.11.2014 04:15">
```

DOM či jiný parser to dá do hajzlu

XML – dokument

› Komentáře

- › Jsou při zpracování ignorovány
- › Nemohou být složeny dovnitř značky

`<match <!-- komentář -->` > tohle celé je komentář

- › Nemohou uvnitř obsahovat sekvenci „--“

› Escapování speciálních znaků

Character	Escape sequence
'	'
"	"
&	&
<	<
>	>

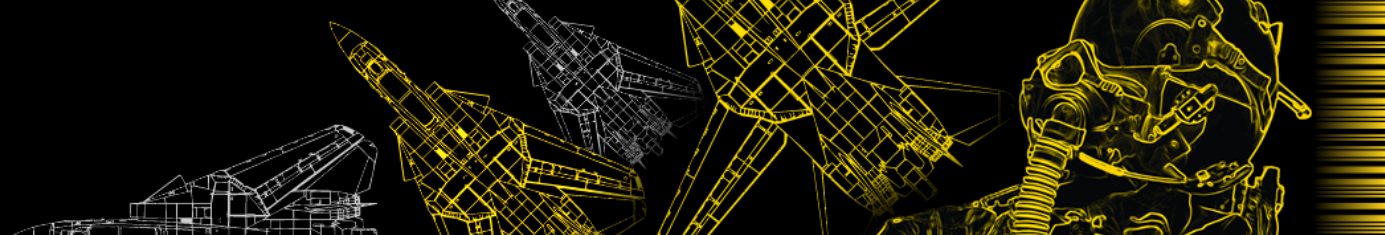
Základní rozdělení přístupu k XML

➤ Kontinuální přístup

- Lze jednoduše popsat tak, že postupně čteme odshora dokument a reagujeme na události načtení jednotlivých částí dokumentu jako např.
 - Počáteční značka elementu, text elementu, koncová značka elementu, počáteční značka elementu, atribut elementu, text elementu, koncová značka elementu
- XML dokument tedy není celý načten do paměti a lze takto zpracovat i „velké“ soubory

➤ Přístup pomocí Document Object Model (DOM)

- Objektová reprezentace stromové struktury XML je načtena do paměti a můžeme se v ní „libovolně pohybovat“.
- Na rozdíl od kontinuálního přístupu nemusí být DOM vhodný pro zpracování „velkých“ souborů
- Důležité třídy pro oba přístupy nalezneme v `System.Xml`



Kontinuální přístup



Kontinuální čtení - parsování

v- nadřazená třída

```
public Auto(string jmenoSouboru) : base() {  
    XmlReader parser = new XmlTextReader(jmenoSouboru);  
    string actualTag = null;  
    bool emptyValue = true;  
  
    while (parser.Read())  
    {  
        switch (parser.NodeType)  
        {  
            case XmlNodeType.Element:  
                actualTag = parser.Name;  
                break;  
  
            case XmlNodeType.Text:  
                ...  
                break;  
  
            case XmlNodeType.EndElement:  
                if (emptyValue)  
                    throw new Exception("Element " + actualTag + " nemá definován obsah");  
                actualTag = null;  
                emptyValue = true;  
                break;  
        }  
    }  
}
```

Kontinuální čtení – načítání atributů

```
public Auto(string jmenoSouboru) : base() {
    double prevodnik = 1;
    ...
    while (parser.Read()) {
        switch (parser.NodeType) {
            case XmlNodeType.Element:
                if (parser.Name.Equals("maxRychlost") && parser.HasAttributes)
                    if (parser["jednotka"].Equals("mil/h"))
                        prevodnik = 1.61;
                    break;
            case XmlNodeType.Text:
                ... this.maxRychlost = Convert.ToInt32(parser.Value) * prevodnik;
                break;
            case XmlNodeType.EndElement:
                ...
                prevodnik = 1;
                break;
        }
    }
}
```

Kontinuální čtení – čtení datových typů

```
public Auto(string jmenoSouboru) : base() {
    XmlReader parser = new XmlTextReader(jmenoSouboru);
    while (parser.Read()) {
        if (parser.NodeType == XmlNodeType.Element) {
            switch (parser.Name) {
                case "znacka":
                    this.name = parser.ReadElementContentAsString();
                    break;
                case "pocetOsob":
                    this.pocetOsob = parser.ReadElementContentAsInt();
                    break;
                case "maxRychlost":
                    double prevodnik = ((parser["jednotka"].Equals("mil/h")) ? 1.61 : 1);
                    this.maxRychlost = parser.ReadElementContentAsDouble() * prevodnik;
                    break;
                case "motory:vykon":
                    motor = new Motor(parser.ReadElementContentAsInt());
                    break;
            }
        }
    }
}
```

Nastavení při vytvoření XmlReader

➤ Metoda Create s předáním XmlReaderSettings

```
XmlReaderSettings settings = new XmlReaderSettings();  
settings.IgnoreComments = true;  
settings.IgnoreProcessingInstructions = true;  
settings.IgnoreWhitespace = true;  
XmlReader parser = XmlReader.Create(kjmenoSouboru, settings);
```

Zápis XML

› Třída XmlWriter

› Vytvoření pomocí metody Create

```
XmlWriter parser = XmlWriter.Create(jmenoSouboru);
```

› Zápis dat

› Hlavička XML

```
writer.WriteStartDocument();
```

› Element

› Zápis rozdělen na zápis počáteční a koncové značky

```
writer.WriteStartElement("prefix", "name", "namespace");  
writer.WriteEndElement();
```

› Hodnota

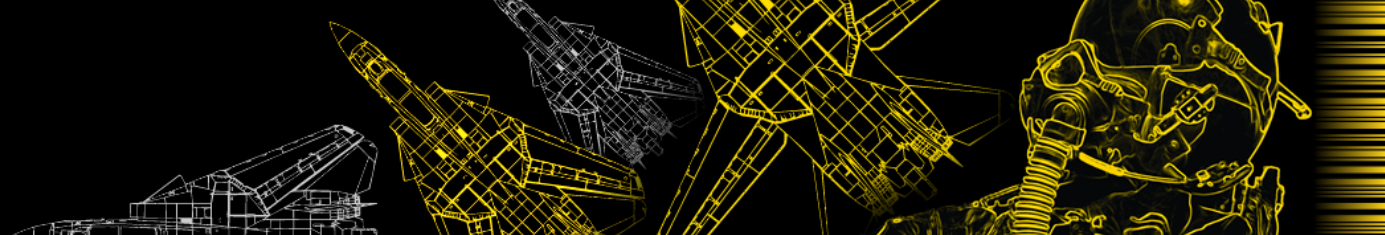
```
writer.WriteValue(object);
```

› Atribut (do aktuálního elementu)

```
writer.WriteAttributeString("jednotka", "km/h");
```


Zápis do XML

```
public void SaveToXMLAuto() {  
    XmlWriter w = null;  
    string autaURI = "http://www.vozovypark.cz/auta.html";  
    string motoryURI = "http://www.vozovypark.cz/motory.html";  
    try {  
        XmlWriterSettings settings = new XmlWriterSettings();  
        settings.Indent = true; settings.OmitXmlDeclaration = false;  
        settings.NewLineOnAttributes = false;  
        w = XmlWriter.Create(new StreamWriter("C:/temp/auto2.xml"), settings);  
        w.WriteStartDocument();  
        w.WriteStartElement("", "auto", autaURI);  
        w.WriteElementString("znacka", this.name);  
        w.WriteElementString("pocetOsob", this.pocetOsob.ToString());  
        w.WriteStartElement("maxRychlost");  
        w.WriteAttributeString("jednotka", "km/h"); w.WriteValue(this.maxRychlost);  
        w.WriteEndElement();  
        w.WriteStartElement("motory", "motor", motoryURI);  
        w.WriteStartElement("vykon", motoryURI);  
        w.WriteAttributeString("jednotka", "kw"); w.WriteValue(this.motor.vykon);  
        w.WriteEndElement(); w.WriteEndElement(); w.WriteEndElement();  
        w.WriteEndDocument();  
    }  
    ...  
    finally { w.Close(); }  
}
```



Document Object Model


The Node object is the primary data type for the entire DOM. A node can be an element node, an attribute node, a text node, or any other of the node types explained in the "Node types" chapter. An XML element is everything from (including) the element's start tag to (including) the element's end tag. 25. 9. 2008



Document Object Model

- Základem práce je třída `XmlNode`
- `XmlNode` je společným předkem tříd pro práci s XML přičemž nejčastěji použitými jsou
 - `XmlDocument` – celý dokument XML
 - `XmlElement` – obecný XML element
 - `XmlAttribute` – atribut elementu
- Načtení XML do struktury **DOM** v paměti

```
XmlDocument doc = new XmlDocument();  
doc.Load(jmenoSouboru);
```



Práce s DOM

➤ Získání kořenového elementu

```
XmlNode root = doc.DocumentElement;
```

➤ Získání potomků elementu

```
XmlNodeList childs = root.ChildNodes;
```

➤ Iterace skrze kolekci elementů

```
foreach (XmlNode node in root.ChildNodes)
{
    ...
}
```

➤ Získání obsahu dopsat InnerText a InnerXml

`node.InnerText` Vrátí text patřící k nodu

`node.InnerXml` Inner XML vyhodí sub-xml, outer.xml i s tím vnějším tagem, na kterém to volám

➤ Získání atributů

```
XmlAttributeCollection attrs = node.Attributes;
```

➤ Získání názvu a hodnoty atributu

`at.Name`

`at.Value`

Ukázka načítání pomocí DOM

```
public Auto(string jmenoSouboru) : base() {
    XmlDocument dok = new XmlDocument(); dok.Load(jmenoSouboru);
    XmlNode root = dok.DocumentElement; XmlNodeList childs = root.ChildNodes;
    foreach (XmlNode node in root.ChildNodes) {
        switch (node.Name) {
            case "znacka":
                this.name = node.InnerText;
                break;
            case "pocetOsob":
                this.pocetOsob = Convert.ToInt32(node.InnerText);
                break;
            case "maxRychlost":
                double prevodnik = 1; XmlAttributeCollection attrs = node.Attributes;
                foreach (XmlAttribute at in attrs)
                    if (at.Name.Equals("jednotka") && at.Value.Equals("mil/h"))
                        prevodnik = 1.61;
                this.maxRychlost = Convert.ToDouble(node.InnerText) * prevodnik;
                break;
            case "motory:motor":
                foreach (XmlNode n in node)
                    if (n.LocalName.Equals("vykon"))
                        this.motor = new Motor(Convert.ToInt32(n.InnerText));
                break;
        }
    }
}
```

Hledání uzlu podle názvu

- Využijeme metodu `GetElementsByTagName(string)` na objektu typu `XmlElement`

```
public Auto(string jmenoSouboru) : base() {  
    XmlDocument dok = new XmlDocument(); dok.Load(jmenoSouboru);  
    XmlNode root = dok.DocumentElement; Node může být i list stromu, tedy i např text na konci, tj element  
                                     je výše a dole se na ten element přetypovává aby nad ním šlo pustit  
                                     getElementsByTagName, node je nadřazená třída od elementu, proto expliciti  
    XmlNodeList znackaLists = ((XmlElement)root).GetElementsByTagName("znacka");  
    this.name = znackaLists[0].InnerText;  
    XmlNodeList osobyLists = ((XmlElement)root).GetElementsByTagName("pocetOsob");  
    this.pocetOsob = Convert.ToInt32(osobyLists[0].InnerText);  
  
    XmlNodeList maxRLists = ((XmlElement)root).GetElementsByTagName("maxRychlost");  
    double prevodnik = 1;  
    string attrString = ((XmlElement) maxRLists[0]).GetAttribute("jednotka");  
    if (attrString != null && attrString.Equals("mil/h"))  
        prevodnik = 1.61;  
    this.maxRychlost = Convert.ToDouble(maxRLists[0].InnerText) * prevodnik;  
  
    XmlNodeList vykonLists = ((XmlElement)root).GetElementsByTagName("motory:vykon");  
    this.motor = new Motor(Convert.ToInt32(vykonLists[0].InnerText));  
}
```

Zápis pomocí DOM

› Typicky dvě situace

- › Modifikace existujícího dokumentu
- › Vytvoření nového dokumentu

› Postup

- › Vytvoření instance XmlDocument
- › (v případě modifikace existujícího dokumentu použijeme metodu Load pro načtení obsahu existujícího dokumentu)
- › Vytvoření/modifikace DOM v paměti pomocí vytváření XmlElement, XmlAttribute atd.
 - › Nezapomínat na přidání na správná místa pomocí metody Append
- › Použití metody Save pro uložení XmlDocument

Ukázka modifikace pomocí DOM

➤ Přidáme element „ridic“ s atributy

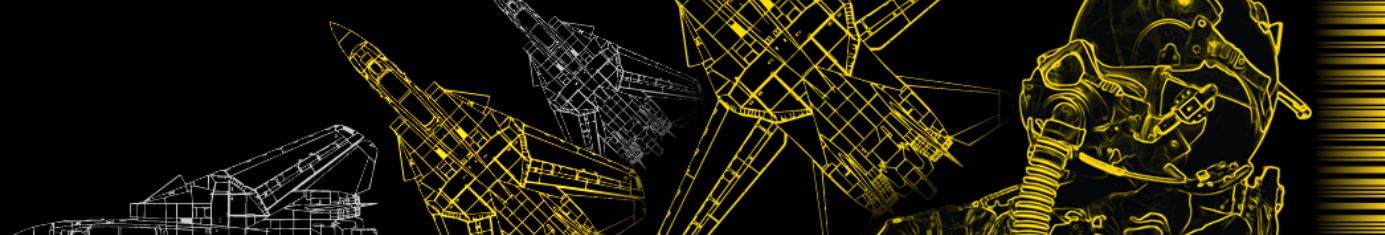
```
public void AddDriverToXML(string jmenoSouboru, Person p)
{
    XmlDocument dok = new XmlDocument();
    dok.Load(jmenoSouboru);
    XmlNode root = dok.DocumentElement;

    XmlElement elem = dok.CreateElement("ridic");

    XmlAttribute nameAttr = dok.CreateAttribute("jmeno");
    XmlAttribute surnameAttr = dok.CreateAttribute("prijmeni");
    XmlAttribute rcAttr = dok.CreateAttribute("rc");
    nameAttr.Value = p.name; surnameAttr.Value = p.surname; rcAttr.Value = p.rc;

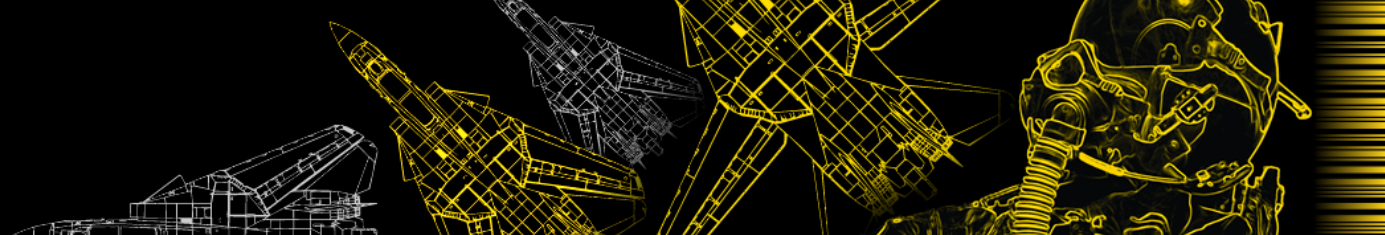
    elem.Attributes.Append(nameAttr);
    elem.Attributes.Append(surnameAttr);
    elem.Attributes.Append(rcAttr);

    root.AppendChild(elem);
    dok.Save(jmenoSouboru);
}
```

Cvičení – Kontinuální přístup a DOM





XSLT



XSLT

➤ Než začneme hovořit o XSLT, popišme co je XSL.

➤ **XSL = Extensible Stylesheet Language** Jazyk jak z jednoho XML udělat jiné XML, nebo HTML.

➤ Vyvinuto W3C jako „Stylesheet Language“ založený na XML

➤ XML nepoužívá předdefinované tagy a tudíž význam jednotlivých elementů nemusí být správně interpretován.

➤ Např. **element <table>** může reprezentovat HTML tabulku nebo **např. kus nábytku.**

➤ XSL tedy popisuje „jak mají být XML elementy zobrazeny“.

➤ XSL je „více než Stylesheet Language“. **Skládá se z následujících částí:**

➤ **XSLT** – jazyk pro transformaci XML dokumentů

➤ **XPath** – jazyk pro navigaci v XML dokumentech

➤ **XSL-FO** – jazyk pro formátování XML dokumentů (od roku 2013 „ukončen“ a jako náhrada doporučeno **CSS3**)

➤ **XQuery** – jazyk pro dotazování v XML dokumentech

XSLT

- XSLT = XSL Transformations
- Typicky používá k transformaci XML dokumentu do
 - Jiného XML dokumentu
 - Dokumentu podporovaného prohlížeči jako např. HTML
- Pomocí XSLT lze přidávat/odebírat elementy a atributy do/z výsledného dokumentu.
- Pro navigaci používá jazyk XPath
- Proces transformace probíhá tak, že pomocí XPath definujeme části zdrojového dokumentu (nemusíme vybrat všechno ze zdrojového dokumentu) a způsob jakým mají být transformovány (např. elementy <row> budou transformovány na elementy <tr>)

Deklarace XSL Stylesheet dokumentu

- Kořenovým elementem by dle W3C XSLT doporučení měl být

```
<?xml version="1.0" encoding="utf-8"?>  
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
</xsl:stylesheet>
```

- Nebo

```
<?xml version="1.0" encoding="utf-8"?>  
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
</xsl:transform>
```

- Dle W3C mají elementy `<xsl:stylesheet>` a `<xsl:transform>` stejný význam. Můžete používat libovolný z nich.
- V obou případech se odkazujeme na oficiální XSLT jmenný prostor W3C.

Ukázka transformace XML do HTML

- Jde pouze o ilustrativní ukázkou na úvod. Význam elementů pro transformaci budeme vysvětlovat dále. Mějme následující XML soubor.
- Všimněte si odkazu na XSL soubor pomocí `<?xml-stylesheet...>`

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  <!-- elementů cd tu bude více ... ->
</catalog>
```

Ukázka transformace XML do HTML

➤ A ke XML souboru mějme následující cdcatalog.xsl soubor

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h2>My CD Collection</h2>
        <table border="1">
          <tr bgcolor="#9acd32">
            <th>Title</th>
            <th>Artist</th>
          </tr>
          <xsl:for-each select="catalog/cd">
            <tr>
              <td>
                <xsl:value-of select="title"/>
              </td>
              <td>
                <xsl:value-of select="artist"/>
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Ukázka transformace XML do HTML

➤ Výsledek transformace bude následující

```
<html>
  <head></head>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tbody>
        <tr bgcolor="#9acd32">
          <th>Title</th>
          <th>Artist</th>
        </tr>
        <tr>
          <td>Empire Burlesque</td>
          <td>Bob Dylan</td>
        </tr>
        ...Elementů <tr> zde bude více
      </tbody>
    </table>
  </body>
</html>
```


Element <xsl:template>

- XSL style sheet se skládá z jedné či více sad pravidel nazývaných šablony (templates).
- Šablonu definujeme pomocí elementu <xsl:template>
 - Pomocí atributu match asociujeme šablonu s příslušným XML elementem. Atribut match může být rovněž použit tak, aby byl asociován s celým XML dokumentem. Např. match="/" asociuje šablonu s celým dokumentem.
 - V atributu match používáme výraz jazyka XPath, který není součástí výkladu v tomto kurzu. Zájemce odkazujeme např. na https://www.w3schools.com/xml/xpath_intro.asp
 - *Poznámka: můžete se vrátit a podívat se na předchozí ukázkou jak byl tento element a atribut použit.*

Element `<xsl:value-of>`

- Element `<xsl:value-of>` slouží k získání hodnoty XML elementu a její vložení do výstupu transformace.
- V atributu `select` definujeme pomocí výrazu v jazyce XPath element jehož obsah má být získán.
- Např. `<xsl:value-of select="catalog/cd/title"/>` vybere obsah prvního elementu `title`, který je prvním potomkem (potomkem tohoto názvu) elementu `cd`, který je prvním potomkem (potomkem tohoto názvu) kořenového elementu `catalog`.

Element <xsl:for-each>

- Element <xsl:for-each> umožňuje implementaci iterace (cyklu) v XSLT.
- Pomocí atributu `select` vybere příslušnou kolekci elementů. Stejně jako v předchozím případě používáme výraz jazyka Xpath.
- *Poznámka: můžete se vrátit a podívat se na ukázkou ze začátku této části o XSLT, kde je tento element použit.*

Element <xsl:sort>

- Pro seřazení výstupu je možné používat tento element.
- Pomocí atributu `select` definujeme výraz v jazyce XPath, kterým vybereme element, podle jehož obsahu bude výstup seřazen.

```
<xsl:for-each select="catalog/cd">
  <xsl:sort select="artist"/>
  <tr>
    <td>
      <xsl:value-of select="title"/>
    </td>
    <td>
      <xsl:value-of select="artist"/>
    </td>
  </tr>
</xsl:for-each>
```

Element <xsl:if>

- Element <xsl:if> umožňuje definovat podmínku a obsah, který má být vložen do výstupu transformace, pokud je podmínka splněna.
- Do atributu test umísťujeme logický výraz (XPath).
- Do obsahu elementu vložíme to, co má být vloženo do výstupu transformace, pokud je hodnota logického výrazu true.
- Element <xsl:if> můžeme použít v kombinaci s <xsl:for-each>

```
<xsl:for-each select="catalog/cd">  
  <xsl:if test="price > 10">  
    <tr>  
      <td><xsl:value-of select="title"/></td>  
      <td><xsl:value-of select="artist"/></td>  
      <td><xsl:value-of select="price"/></td>  
    </tr>  
  </xsl:if>  
</xsl:for-each>
```

Element <xsl:choose>

- Analogie s příkazem switch ze C#. Princip použití je podobný jako v případě <xsl:if>

```
<xsl:for-each select="catalog/cd">
  <tr>
    <td><xsl:value-of select="title"/></td>
    <xsl:choose>
      <xsl:when test="price > 10">
        <td bgcolor="#ff00ff">
          <xsl:value-of select="artist"/></td>
        </xsl:when>
      <xsl:when test="price > 9">
        <td bgcolor="#cccccc">
          <xsl:value-of select="artist"/></td>
        </xsl:when>
      <xsl:otherwise>
        <td><xsl:value-of select="artist"/></td>
      </xsl:otherwise>
    </xsl:choose>
  </tr>
</xsl:for-each>
```

Element <xsl:apply-templates>

- Tento element se používá k aplikaci šablony na příslušný element nebo na jeho potomky. Pokud v tomto elementu definujeme atribut `select`, pak budou zpracovány pouze odpovídající potomci.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h2>My CD Collection</h2>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="cd">
    <p>
      <xsl:apply-templates select="title"/>
      <xsl:apply-templates select="artist"/>
    </p>
  </xsl:template>
  <xsl:template match="title">
    Title: <span style="color:#ff0000">
      <xsl:value-of select="."/></span>
    <br />
  </xsl:template>
  <xsl:template match="artist">
    Artist: <span style="color:#00ff00">
      <xsl:value-of select="."/></span>
    <br />
  </xsl:template>
</xsl:stylesheet>
```

XSLT v .NET

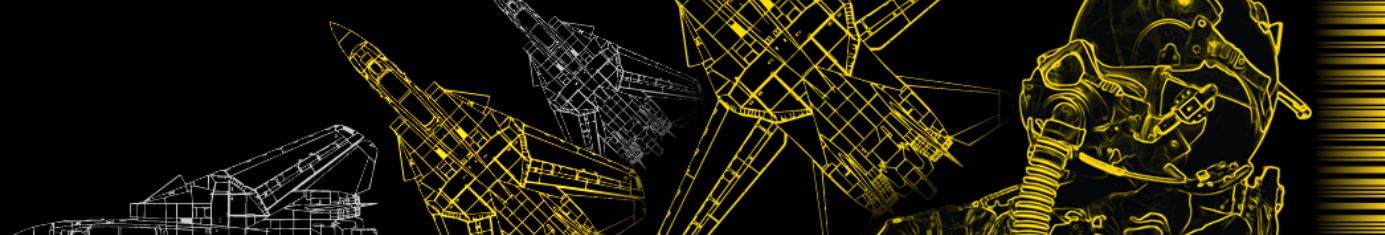
- Ukázka provedení transformace s použitím třídy `System.Xml.Xsl.XslCompiledTransform`
- Pro více informací vizte [MSDN - XSLT Transformations](#)

```
using System.Xml;
using System.Xml.Xsl;
namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            XmlReader reader = XmlReader.Create("cdcatalog.xml");
            XmlWriter writer = XmlWriter.Create("cdcatalog.html");

            XslCompiledTransform transform = new XslCompiledTransform();
            XsltSettings settings = new XsltSettings();
            settings.EnableScript = true;

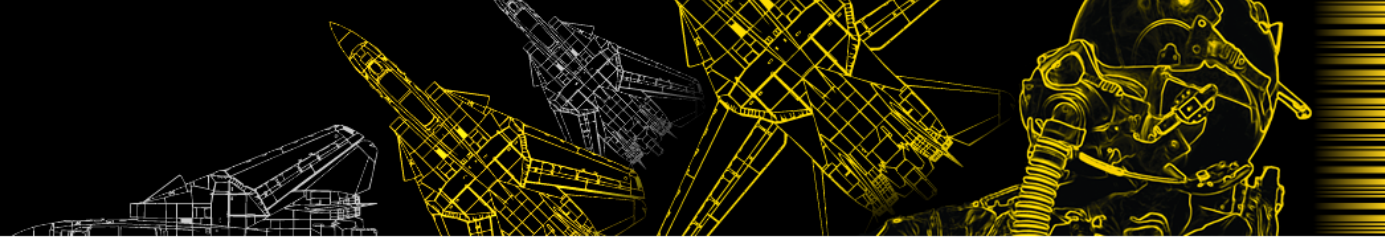
            transform.Load("cdcatalog.xsl", settings, null);

            transform.Transform(reader, writer);
        }
    }
}
```

Cvičení - XSLT





XML Schema (XSD)



XML Schema

- *Cílem není naučit se tvořit XML schémata, ale udělat si představu o jejich principech a zkusit si jejich použití v .NET.*
- Používá se i název XML Schema Definition (XSD)
- Popisuje strukturu dokumentu.
- Definuje použitelné stavební bloky pro XML dokument:
 - Elementy a atributy, které se mohou vyskytovat v dokumentu
 - Počet (a pořadí) potomků
 - Datové typy elementů a atributů
 - Výchozí a pevně dané hodnoty elementů a atributů
- Založeno na XML.
- „Mocnější alternativa k DTD“.

XML Schema - Ukázka

➤ XML Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

➤ Odpovídající XML dokument

```
<?xml version="1.0"?>
<note
  xmlns="https://www.w3schools.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://www.w3schools.com/xml/note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

z tohoto jmeného prostoru se berou například formáty stringů,
datumů



Element <schema>

- Kořenový element pro každé XML Schema.
- Typicky obsahuje následující (může obsahovat i jiné) atributy:
 - `xmlns:xs="http://www.w3.org/2001/XMLSchema"`
 - V dokumentu budeme používat elementy a datové typy z daného jmenného prostoru
 - `targetNamespace="TARGET_NAMESPACE"`
 - Elementy definované tímto schématem pochází z daného jmenného prostoru.
 - `xmlns="DEFAULT_NAMESPACE"`
 - Výchozí jmenný prostor pro tento dokument.
 - `elementFormDefault="qualified"`
 - Každý dokument, ve kterém budeme toto schéma referencovat musí používat jmenným prostorem kvalifikované názvy elementů.

Reference na XML Schema z Dokumentu

```
<?xml version="1.0"?>
<note xmlns="https://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- `xmlns="https://www.w3schools.com"`
 - výchozí jmenný prostor pro dokument
- `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
`xsi:schemaLocation="https://www.w3schools.com note.xsd"`
 - specifikujeme umístění XML schématu pro daný jmenný prostor

XML Schema – Simple Elements

- XML element, který obsahuje pouze „text“.
- Nemůže obsahovat elementy ani atributy.
- Můžeme specifikovat typ „textu“ v elementu.
- Typicky používané vestavěné typy:
 - xs:string
 - xs:decimal
 - xs:integer
 - xs:boolean
 - xs:date
 - xs:time
- Lze používat i výchozí a fixní hodnoty (jiná hodnota nelze v elementu nastavit).
- Ukázka:

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
<xs:element name="color" type="xs:string" default="red"/>
<xs:element name="color" type="xs:string" fixed="red"/>
```

XML Schema - atributy

- Simple elementy nemohou obsahovat atributy. Pokud by element obsahoval atributy jde o tzv. complex type (o těch bude řeč za chvíli).
- Atributy lze v XML Schema definovat analogicky jako elementy (včetně omezení datových typů):

```
<xs:attribute name="xxx" type="yyy"/>
```

- Lze rovněž používat výchozí a fixní hodnoty:

```
<xs:attribute name="lang" type="xs:string" default="EN"/>
```

```
<xs:attribute name="lang" type="xs:string" fixed="EN"/>
```


XML Schema – Complex Types

- Elementy, které mohou obsahovat jiné elementy nebo atributy.
- Dva způsoby definice:

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```



```
<xs:element name="employee" type="personinfo"/>
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

- Poznámka: <sequence> určuje, že elementy musí být v daném pořadí. Pomocí atributu maxOccurs můžeme specifikovat maximální možný počet opakování elementu (např. unbounded).

XML Schema – Validace v .NET

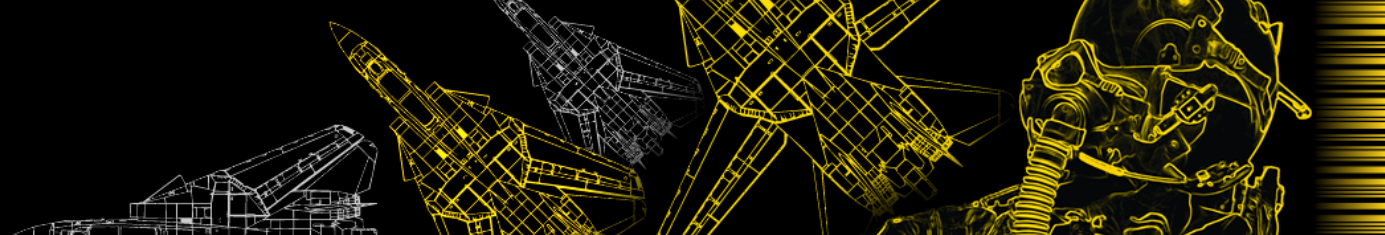
- `XmlReader.Create()`
 - Parametrem lze předat `XmlReaderSettings`
 - `Schemas` – seznam schémat
 - `ValidationType` – nastavíme, že budeme validovat proti XSD (podporuje i None a DTD)
 - `ValidationEventHandler` – obsluha události, ve které můžeme reagovat na chyby při validaci dokumentu
 - `ValidationEventArgs.Severity` (Warning, Error)

XML Schema – Validace v .NET - Ukázka

```
static void Main()
{
    XmlReaderSettings booksSettings = new XmlReaderSettings();
    booksSettings.Schemas.Add("http://www.unicorn.eu/hatchery", "XMLSchema1.xsd");
    booksSettings.ValidationType = ValidationType.Schema; XML budem validovat proti XSD schématu;
    booksSettings.ValidationEventHandler += new ValidationEventHandler(booksSettingsValidationEventHandler);
    poslední řádek předá metodu, která se zavolá, když validace neprojde !!
    XmlReader books = XmlReader.Create("bookstore.xml", booksSettings);

    while (books.Read()) { }
}

static void booksSettingsValidationEventHandler(object sender, ValidationEventArgs e)
{
    if (e.Severity == XmlSeverityType.Warning)
    {
        Console.Write("WARNING: ");
        Console.WriteLine(e.Message);
    }
    else if (e.Severity == XmlSeverityType.Error)
    {
        Console.Write("ERROR: ");
        Console.WriteLine(e.Message);
    }
}
```



Cvičení - XSD



UNICORN

TOP GUN®

ACADEMY



www.unicorn.eu