

XSLT (eXtensible StyleSheet Language for Transformations)



Преглед на XSL
Употреба
Възможности
XSLT елементи
Шаблони
Манипулиране
Примери

Стилови множества (Style Sheet)

- ***CSS - Cascading Style Sheet Specification***

- Предоставя прост синтаксис за добавяне на стилове към елементи (в HTML браузъри)

- ***DSSSL - Document Style and Semantics Specification Language***

- Международен SGML стандарт за стилове и конвертиране на документи

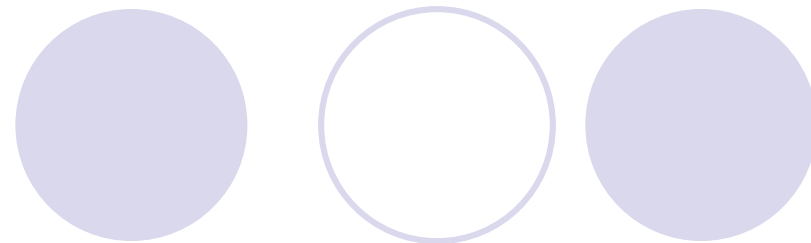
- ***XSL - Extensible Style Language***

- Комбиниращи черти на DSSSL и CSS, използвайки XML синтаксис

Какво е XSL?

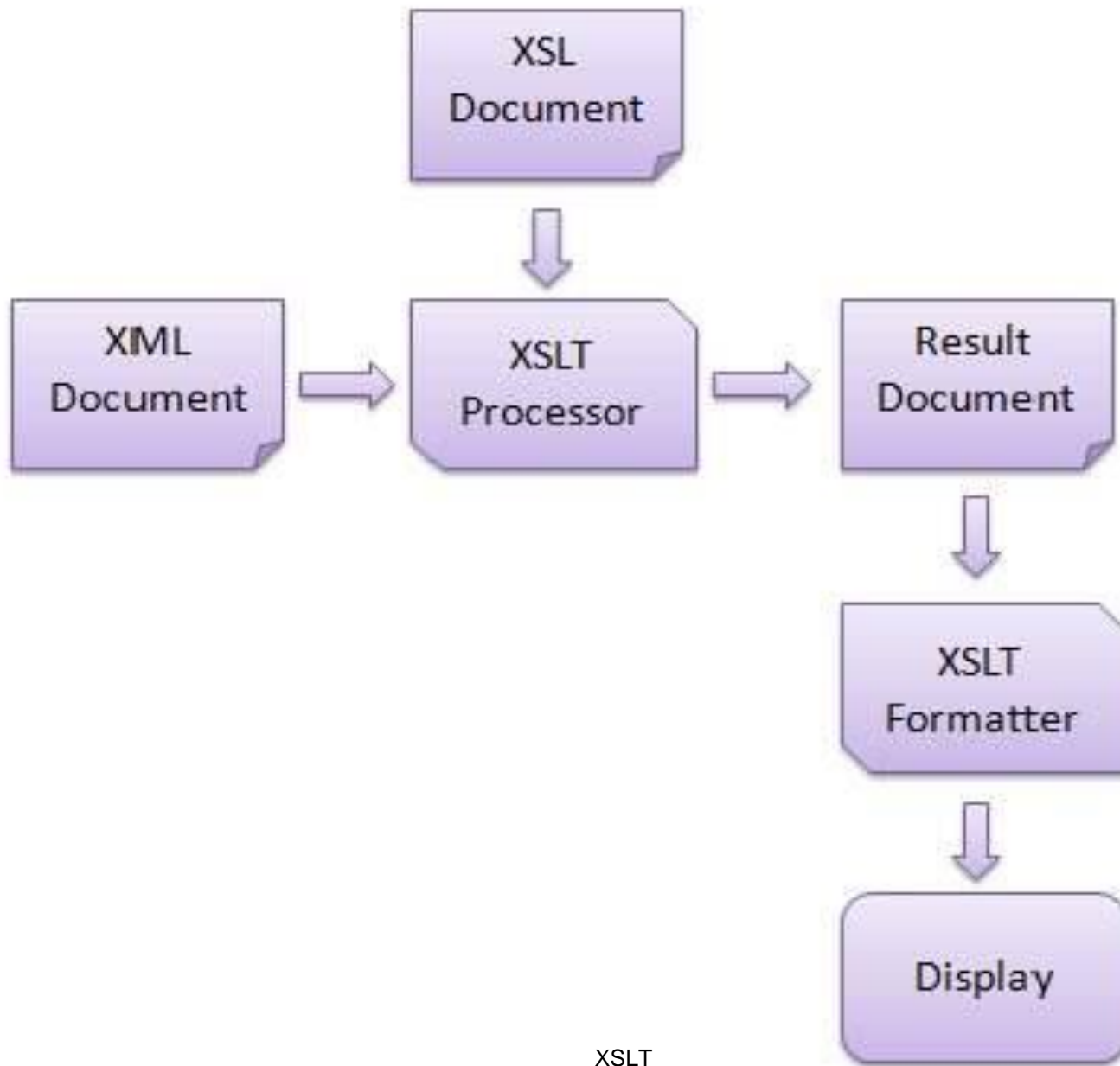
- **XSL** = e**X**tensible **S**tylesheet **L**anguage
 - Трансформира XML до друг текстов формат, напр.
 - XML
 - HTML
 - Text
 - Притежава т.нар. "scripting" engine – XSLT
 - За разлика от императивни езици като JavaScript, XSL е *декларативен* => без странични ефекти

XSL-FO и XSLT



XSL се състои от две части:

- **Extensible Stylesheet Language – Formatting Objects (XSL-FO)** – език за описание на форматирането на данните в XML документ с цел представянето им на различни медии (напр. екран, принтер или мултимедия);
- **Extensible Stylesheet Language for Transformation (XSLT)** – за трансформиране на XML документи с помощта на различни стилове и функции. Най-често се използва за конвертиране на документ от XML формат към документ в HTML формат, обикновен текстов файл или пък например друг XML документ. Полезен е, когато искаме да разделим презентационния слой на едно приложение от модела на данните му



Предназначение на XSLT

Преобразуване на XML в специфичен формат за **представяне**, като например HTML

Преобразуване от формат, **разбираем за едно приложение**, във формат, **разбираем за друго приложение**

XSLT като декларативен език 1-1

- Процедурен език
 - Софтуерът получава данни и ги обработва стъпка по стъпка
 - Всеки израз или блок изпълнява точно дефинирана задача, реализирайки промени в данните
- Пример: визуализиране на имена в колекция Author

Java

```
int index;  
for (index = 0; index < allAuthors.Count; index++)  
{  
    Author thisAuthor = allAuthors[index];  
    Console.WriteLine(thisAuthor.FirstName + " " + thisAuthor.LastName);  
}
```

C#

```
foreach (Author thisAuthor in allAuthors)  
{  
    Console.WriteLine(thisAuthor.FirstName + " " +  
        thisAuthor.LastName);  
}
```

SQL

```
SELECT FirstName, LastName FROM Authors;
```



Процедурни
езици

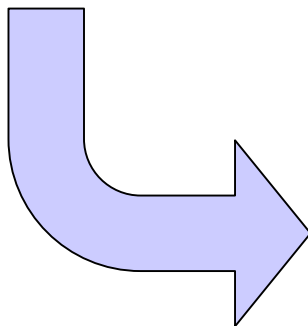


Декларативни
езици

XSLT като декларативен език 2-2

- Липса на необходимост от деклариране на променлива, с която да се управлява итерацията на елементите в колекцията
- XSLT шаблон

```
<xsl:template match="author" />  
  <xsl:value-of select="firstName" />  
  <xsl:value-of select="lastName" />  
</xsl:template>
```



```
<authors>  
  <author>  
    <firstName>Danny</firstName>  
    <lastName>Ayers</lastName>  
  </author>  
  <author>  
    <firstName>Joe</firstName>  
    <lastName>Fawcett</lastName>  
  </author>  
  ...  
</authors>
```


XSLT като функционален език

- Императивен език

- Java, C++, C# и PHP диктуват какво точно трябва да се направи, опитвайки да променят състоянието на обекта, за да представят промените в обстоятелствата

- Пример: промяна на елемент в колекция Author

```
Author authorToEdit = getAuthor(12345);  
//Get the required author using their ID  
authorToEdit.LastName = "Marlowe"; //Change last name
```

- Функционален език

- Изходът е резултат от прилагане на една или повече функции върху входа

Конфигуриране на локална XSLT среда за разработка

- Saxon процесор
 - Поддържа актуална реализация на XSLT
 - Предлага се с отворен достъп (комерсиалните версии са с по-богата функционалност)
 - Предоставя се с версии за Java и .NET
- Уеб сайт: <http://saxon.sourceforge.net/>
- Инсталиране за .NET
 - Конфигуриране на PATH променливата на средата (c:\Program Files\Saxonica\SaxonHE9.3N\bin)
 - Отпада необходимост от специфициране на пълния път до файловете в командния ред при трансформация
 - Команден ред: Transform -?
- Инсталиране за Java
 - Изисква JVM
 - Конфигуриране на CLASSPATH променливата на средата (<installation path>/saxon9he.jar)
 - Команден ред: java net.sf.saxon.Transform -?
- Документация: <http://www.saxonica.com/documentation/>

Възможности на XSL/XSLT

- Добавяне на префиксен или суфиксен текст към съществуващо съдържание
- Структурни промени на входното съдържание, като създаване, отстраняване, редактиране, пренареждане и сортиране на XML елементи
- Многократно използване на елементно или атрибутно съдържание на друго място в документа
- Трансформиране на данни между XML формати
- Определяне на XSL форматиращи обекти и на други средства за представяне на съдържанието в дадена медиа (напр. CSS), с цел да се прилагат към даден елемент

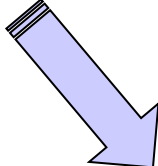
Базови XSLT елементи

- `<xsl:stylesheet>`
 - Задава обхват на XSLT документа и осигурява конфигуриране на параметри
- `<xsl:template>`
 - Специфицира кои елементи трябва да бъдат обхванати от обработката
 - Дефинира какво ще бъде добавено в изходния документ
- `<xsl:apply-templates>`
 - Взима решение кои елементи ще бъдат обработени
- `<xsl:value-of>`
 - Изчислява израз и добавя резултата в изхода
- `<xsl:for-each>`
 - Осигурява групова обработка на елементи по еднотипен начин

Прост пример

- Файл data.xml:

```
<?xml version="1.0"?>  
  <?xml-stylesheet type="text/xsl" href="render.xsl"?>  
  <message>Howdy!</message>
```



- Файл с дефиниция на трансформация render.xsl:

```
<?xml version="1.0"?>  
  <xsl:stylesheet version="1.0"  
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
    <!-- one rule, to transform the input root (/) -->  
    <xsl:template match="/">  
      <html><body>  
        <h1><xsl:value-of select="message"/></h1>  
      </body></html>  
    </xsl:template>  
  </xsl:stylesheet>
```

Файл с разширение .xsl

- Всеки XSLT документ има .xsl разширение

- XSLT документа започва с:

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

`<xsl:stylesheet version="1.0"`

`xmlns:xsl="http://www.w3.org/1999/
XSL/Transform">`

- Може да съдържа шаблони, като напр.:

`<xsl:template match="/"> ... </xsl:template>`

- Завършва с:

`</xsl:stylesheet>`

Важно:

Елементите `<xsl:stylesheet>` и `<xsl:transform>` са синоними и могат да се използват взаимозаменяемо.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

Без xsl трансформация...

```
▼<person>
  ▼<name>
    <title>Bai</title>
    <first>Ganyo</first>
    <middle>Son of his Father</middle>
    <last>Balkanski</last>
  </name>
  <profession>Trader, Image Maker, Politician</profession>
  <position>President of Dirty Business UnLtd.</position>
  ▼<résumé>
    ▼<html>
      ▼<head>
        <title>Resume of G. Balkanski</title>
      </head>
      ▼<body>
        <h1>Ganyo Balkanski</h1>
        ▼<p>
          Who didn't know about Bai Ganyo, who has not heard of him?
        </p>
      </body>
    </html>
  </résumé>
</person>
```

Намиране на текста message

- Шаблонът `<xsl:template match="/">` задава селектиране на целия входен документ, т.е. на *root възела на XML дървото*
- Вместо това,
 - `<xsl:value-of select="message"/>` селектира преките наследници на message
 - Това са XPath изрази, както и аналогичните им:
 - `./message`
 - `./message/text()` (`text()` е **XPath функция**)
- Всеки *XSL stylesheet трябва да бъде добре конструиран XML документ*

Как работи ?

- XSL дефиницията е:

```
<xsl:template match="/">  
  <html><body>  
    <h1><xsl:value-of select="message"/></h1>  
  </body></html>  
</xsl:template>
```
- Шаблонът `<xsl:template match="/">` избира корена
- `<html><body>` `<h1>` се записва в изходния файл
- Съдържанието на **message** се записва в изходния файл
- `</h1>` `</body></html>` се записва в изходния файл
- Резултатният файл е:

```
<html><body>  
  <h1>Howdy!</h1>  
</body></html>
```

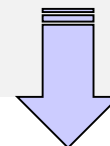
Задаване на XSL трансформация в XML документ

- `<?xml-stylesheet type="text/xsl" href="stylesheet.xsl"?>`
- Тази инструкция задава на брауъра да извика XSLT процесор за изпълнение на XSL трансформацията, зададена в документа със стилове **stylesheet.xsl**, който в случая е от тип **text/xsl**, но би могъл да бъде и от друг тип, напр. **text/css**.

Елемент <xsl:output>

- Позволява контрол над изходния документ
- Разполага се веднага след <xsl:stylesheet>
- Синтаксис:

```
<xsl:output method="xml or html or text"
  version="version"
  encoding="encoding"
  omit-xml-declaration="yes or no"
  standalone="yes or no"
  cdata-section-elements="CDATA sections"
  indent="yes or no" />
```



<http://www.xmlplease.com/xml/standalone/>

Тук задаваме всички елементи, които ще имат CDATA съдържания, като напр.

<elem><![CDATA[<&>&]]></elem>

Елемент <xsl:template>

- Изпълнява кода в шаблона всеки път, когато срещне елемент, който съответства на специфицирания в атрибута `match`
- `match="/"`
 - Шаблонът се изпълнява, когато се срещне кореновият елемент на документа
 - Използва като контекст кореновия елемент

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <!-- basic output here -->
  </xsl:template>
</xsl:stylesheet>
```

Търсене на елемент с XSLT

- `<xsl:template match="exon">`
We have found the EXON tag!
`</xsl:template>`

Шаблон, дефиниран
чрез елемента
`<xsl:template>`

Атрибутът `<match>`
задава шаблон върху
входното дърво

Шаблонът ще се приложи над
всеки възел **exon** във входното
дърво.

Елемент <xsl:value-of>

- Извличане на информация от елементи от началното дърво става чрез елемента
- **<xsl:value-of select="XPath-expression"/>**
- Атрибутът **select='.'** избира контекстния възел и всички негови наследници (за разлика от функцията text() !!!)
- Пример:
- **<xsl:template match="exon">**
- **<xsl:value-of select="."/>**
We have displayed the contents of the EXON tag!
- **</xsl:template>**

```
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="example6.3.2.3.xsl"?>
<persons >
  <person id="123456" age="53">
    <name title="Bai">
      <first>Ganyo</first>
      <last>Balkanski</last>
    </name>
    <profession>Trader, Image Maker, Politician</profession>
    <descr>
```

Who didn't know about Bai Ganyo,
who has not heard of him? ...

```
</descr>
  </person>
  <person id="345" age="29">
    <name title="Mr.">
      <first>Balkan</first>
      <last>Ganyov</last>
    </name>
    <profession>Facilitator</profession>
    <descr>
```

Facilitator of people
who are in the middle of nowhere...

```
</descr>
  </person>
</persons>
```

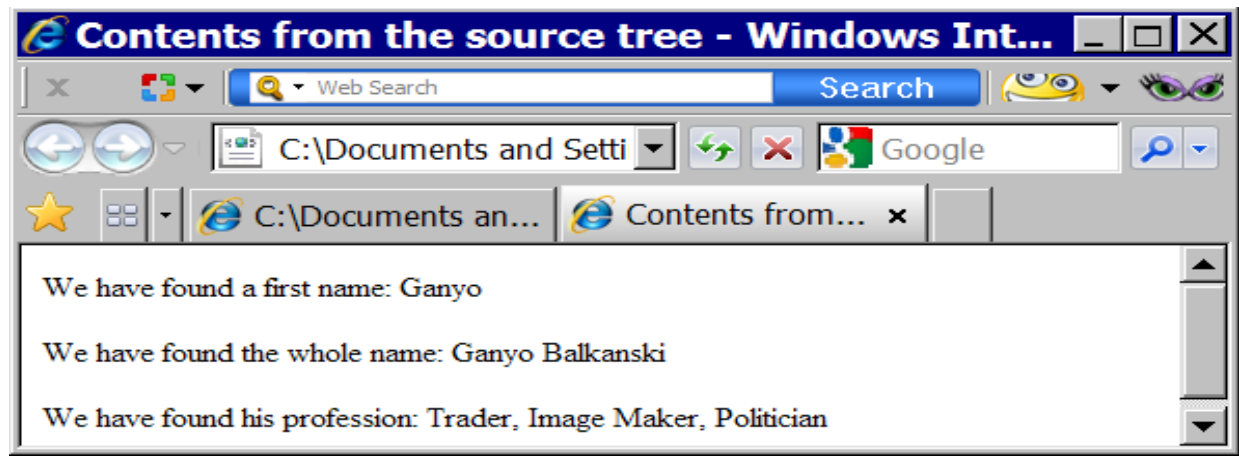
XML

XSLT

За този
ДОКУМЕНТ...

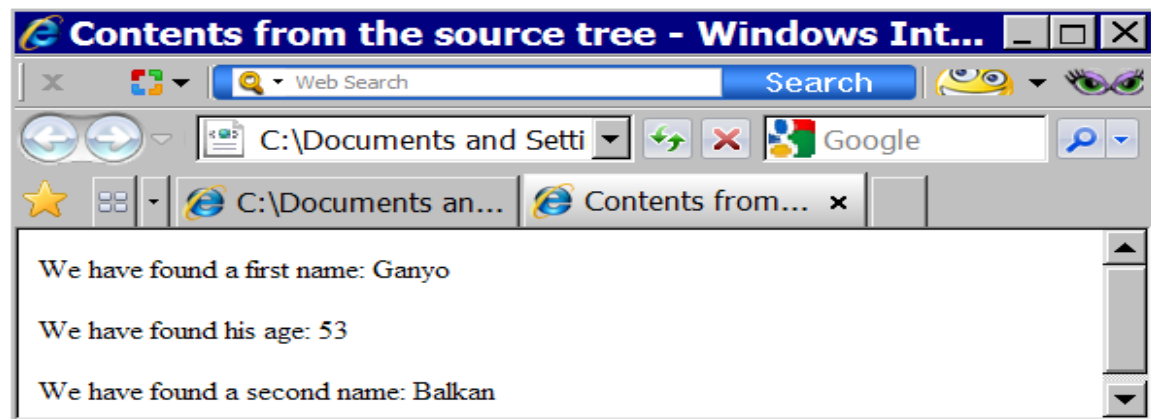
...Трансформация без шаблони

- `<xsl:stylesheet version="1.0"`
- `xmlns:xsl="http://www.w3.org/1999/XSL/Transform">`
- `<html>`
- `<head>`
- `<title>Contents from the source tree</title>`
- `</head>`
- `<body>`
- `<p>We have found a first name:`
- `<xsl:value-of select="/persons/person/name/first"/></p>`
- `<p>We have found the whole name:`
- `<xsl:value-of select="/persons/person/name/."/></p>`
- `<p>We have found his profession:`
- `<xsl:value-of select="/persons/person/profession"/></p>`
- `</body>`
- `</html>`
- `</xsl:stylesheet>`



Търсене на атрибут

- `<xsl:stylesheet version="1.0"`
- `xmlns:xsl="http://www.w3.org/1999/XSL/Transform">`
- `<xsl:template match="/">`
- `<html>`
- `<head>`
- `<title>Contents from the source tree</title>`
- `</head>`
- `<body>`
- `<p>We have found a first name:`
- `<xsl:value-of select="/persons/person/name/first"/></p>`
- `<p>We have found his age:`
- `<xsl:value-of select="/persons/person/@age"/></p>`
- `<p>We have found a second name:`
- `<xsl:value-of select="/persons/person[@age='29']/name/first"/></p>`
- `</body>`
- `</html>`
- `</xsl:template>`
- `</xsl:stylesheet>`



Друг XML пример

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

```
<People>
```

```
<Person diedDate="1965-01-24" bornDate="1874-11-30">
```

```
<Name>Winston Churchill</Name>
```

<Description> Winston Churchill was a mid 20th century British politician who became famous as Prime Minister during the Second World War.

```
</Description>
```

```
</Person>
```

```
<Person diedDate="1984-10-31" bornDate="1917-11-19">
```

```
<Name>Indira Gandhi</Name>
```

<Description> Indira Gandhi was India's first female prime minister and was assassinated in 1984. </Description>

```
</Person>
```

```
<Person diedDate="1963-11-22" bornDate="1917-05-29">
```

```
<Name>John F. Kennedy</Name>
```

<Description> JFK, as he was affectionately known, was a United States president who was assassinated in Dallas, Texas. </Description>

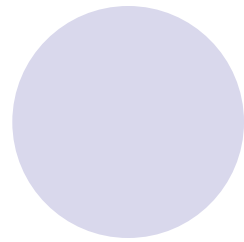
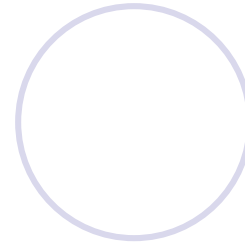
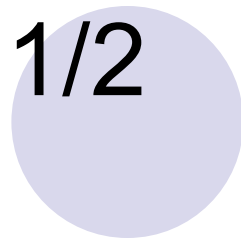
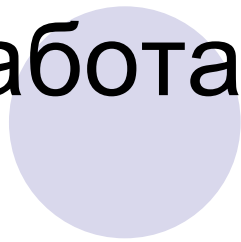
```
</Person>
```

```
</People>
```

Роля на контекста в XSLT

- XSLT обработката винаги се извършва в контекста на определен елемент
- `match="People/Person"`
 - Връща като резултат 3 `<Person>` елемента
 - Селектирането започва от кореновия елемент, преминава през `People` елемента и завършва в `Person` елемента
- `match="Person"`
 - Не връща резултат
 - Селектирането започва от кореновия елемент и спира поради липсата на `Person` елемент
- `match="People"`
 - Връща като резултат 1 `<People>` елемент
 - Селектирането започва от кореновия елемент и завършва в `People` елемента

Работа с шаблони 1/2



- шаблоните в XSLT се дефинират с елемента **<xsl:template>**, чийто атрибут **match** задава образец за съответствие на част от йерархията на началното дърво.
- Стойността на образца представлява XPath израз и се използва за адресиране на секции от началното дърво.

Работа с шаблони 2/2

- Освен параметъра **match**, един шаблон може да има и следните по-важни параметри:
 - ✓ **name** – задава QName, с цел извикването на такъв именован шаблон по име чрез **call-template**;
 - ✓ **priority** – задава числов приоритет, който да се ползва за разрешаване на конфликтите при вземане на решение кой от няколко възможни шаблона да се ползва за даден възел от дървото. Стойността му по подразбиране е 0.5;
 - ✓ **mode** – дава възможност даден възел да бъде обработван многократно, с различен краен резултат в зависимост от стойността на **mode**.

Извикване на шаблони 1/2

- За извикване на шаблони се използва XSLT инструкцията
- **<xsl:apply-templates select="XPath expression" />**
- Активира рекурсивна обработка на всички наследници на контекстния възел
- Пример:

- **<xsl:stylesheet version="1.0"**
- **xmlns:xsl="http://www.w3.org/1999/XSL/Transform">**
- **<xsl:template match="/persons">**
- **<html>**
- **<head>**
- **<title>Found**
- **<xsl:value-of select="count(person)" />**
- **people.**
- **</title>**
- **</head>**

Извикване на шаблони 2/2

- `<body>`
- `<h3>The source XML document contains information about`
- `<xsl:value-of select="count(person)" /> people.</h3>`
- `
`
- `<xsl:apply-templates select="person" />`
- `</body>`
- `</html>`
- `</xsl:template>`
- `<xsl:template match="person">`
- `<h3> <xsl:value-of select="name" />`
- `<i> - professional`
- `<xsl:value-of select="profession" />.</i></h3>`
- `<p><xsl:value-of select="descr" /></p>
`
- `</xsl:template>`
- `</xsl:stylesheet>`

```

<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="example6.3.2.3.xsl"?>
<persons >
  <person id="123456" age="53">
    <name title="Bai">
      <first>Ganyo</first>
      <last>Balkanski</last>
    </name>
    <profession>Trader, Image Maker, Politician</profession>
    <descr>

```

Who didn't know about Bai Ganyo,
who has not heard of him? ...

```

</descr>
</person>
<person id="345" age="29">
  <name title="Mr.">
    <first>Balkan</first>
    <last>Ganyov</last>
  </name>
  <profession>Facilitator</profession>
  <descr>

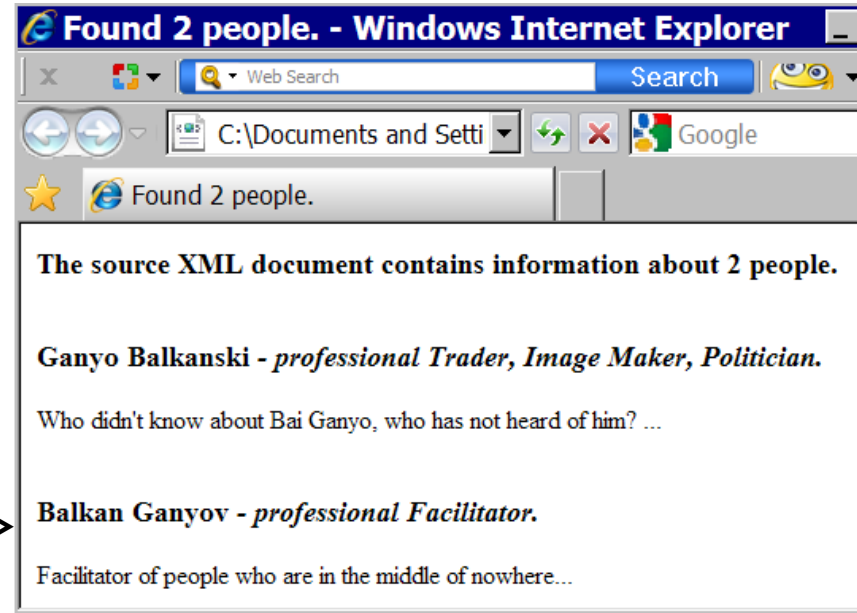
```

Facilitator of people
who are in the middle of nowhere...

```

</descr>
</person>
</persons>

```



Элемент <xsl:apply-templates>

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head>
        <title>Famous People</title>
      </head>
      <body>
        <h1>Famous People</h1>
        <hr />
        ...
      </body>
    </html>
  </xsl:template>
  ...
</xsl:stylesheet>
```



```
<ul>
  <xsl:apply-templates
select="People/Person" />
</ul>
```

Apply-Templates элемент – пример 1

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

```
<simple>
```

```
<name>John</name>
```

```
<name>David</name>
```

```
<name>Andrea</name>
```

```
<name>Ily</name>
```

```
<name>Chaulene</name>
```

```
<name>Cheryl</name>
```

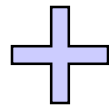
```
<name>Shurnette</name>
```

```
<name>Mark</name>
```

```
<name>Carolyn</name>
```

```
<name>Agatha</name>
```

```
</simple>
```



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

```
<xsl:stylesheet version="1.0"
```

```
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<xsl:template match="/">
```

```
<html>
```

```
<head><title>Sample XSLT Stylesheet </title>
```

```
</head>
```

```
<body>
```

```
<xsl:apply-templates/>
```

```
</body>
```

```
</html>
```

```
</xsl:template>
```

```
<xsl:template match="name">
```

```
<p>Name encountered</p>
```

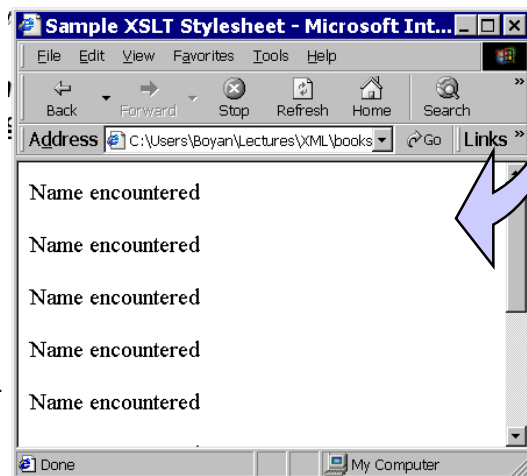
```
</xsl:template>
```

```
</xsl:stylesheet>
```

XSLT in Details



XML



Apply-Templates элемент – пример 2

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

```
<simple>
```

```
<name>John</name>
```

```
<name>David</name>
```

```
<name>Andrea</name>
```

```
<name>Ify</name>
```

```
<name>Chaulene</name>
```

```
<name>Cheryl</name>
```

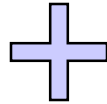
```
<name>Shurnette</name>
```

```
<name>Mark</name>
```

```
<name>Carolyn</name>
```

```
<name>Agatha</name>
```

```
</simple>
```



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

```
<xsl:stylesheet version="1.0"
```

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<xsl:template match="/">
```

```
<html>
```

```
<head><title>Sample XSLT Stylesheet </title>
</head>
```

```
<body>
```

```
<xsl:apply-templates/>
```

```
</body>
```

```
</html>
```

```
</xsl:template>
```

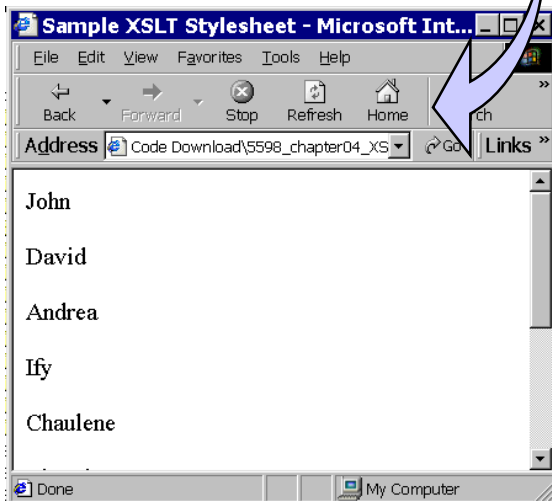
```
<xsl:template match="name">
```

```
<p><xsl:value-of select="."> </p>
```

```
</xsl:template>
```

```
</xsl:stylesheet>
```

XSLT in Details



XML

Работа с шаблони

- По подобен начин шаблонът
- `<xsl:template match="./name">`
- `<xsl:apply-templates select="name[@title='Mr.']/>`
- `</xsl:template>`
- адресира наследници на контекстния възел '.' с име **name** (изразите `"./name"` и `"name"` са еквивалентни)
задава прилагане на най-подходящия шаблон за всички възли с име **name** и с атрибут **title** имащ стойността **'Mr.'**.

Mode атрибут

- Част от елемента template; разрешава извикване на избран шаблон.
- Определя кое match правило да се използва

○ `<xsl:template match="chapter/title">
 <html:h1><xsl:apply-templates/></html:h1>
</xsl:template>`

`<xsl:template match="chapter/title" mode="h3">
 <html:h3><xsl:apply-templates/></html:h3>
</xsl:template>`

`<xsl:template match="intro">
 <xsl:apply-templates
 select="//chapter/title" mode="h3" />
</xsl:template>`

This template is *NOT instantiated* by apply-templates though the MATCH is the same!!

This template is instantiated by apply-templates

Именувани шаблони 1/4

- Осигуряват елементарна обработка
- Атрибут **name**

```
<xsl:template name="iso8601DateToDisplayDate">
  <xsl:param name="iso8601Date" />
  <xsl:variable name="yearPart" select="substring($iso8601Date,1,4)" />
  <xsl:variable name="monthPart" select="substring($iso8601Date,6,2)" />
  <xsl:variable name="datePart" select="substring($iso8601Date,9,2)" />
  <xsl:value-of select="concat($datePart, '/', $monthPart, '/',
$yearPart)" />
</xsl:template>
```

YYYY-
MM-DD

DD/MM/
YYYY

Именувани шаблони 2/4

- Елемент `<xsl:param>`
 - Осигуряват подаване на стойности към шаблона
- Елемент `<xsl:variable>`
 - Променлива, която може да бъде инициализирана еднократно през жизнения си цикъл
- Функция XPath `substring`
 - `substring($iso8601Date, 6, 2)`
- Функция XPath `concat`
 - `concat($datePart, '/', $monthPart, '/', $yearPart)`

Именувани шаблони 3/4

- Инициализиране на променлива

- Атрибут select

```
<xsl:variable name="yearPart" select="..." />
```

- Инициализация със съдържание

```
<xsl:variable name="myVariable">
```

```
  <myElement>Some content</myElement>
```

```
</xsl:variable>
```

- Изисква създаване на ново дърво и добавяне на външен възел

- Достъп до променлива

- Използва се знакът \$

Именувани шаблони 4/4

- Обхват на променлива

- Определя се в зависимост от елемента, в който е дефинирана

```
<xsl:template name="usingVariables">
  <xsl:for-each select="someElements/someElement">
    <xsl:variable name="demo" select="'Some text'" />
    <!-- the line is okay as $demo is in scope -->
    <xsl:value-of select="concat(someElement, $demo)" />
  </xsl:for-each>
  <!--the line is an error as $demo is out of scope-->
  <xsl:value-of select="$demo" />
</xsl:template>
```

Елемент <xsl:call-template>

- Елемент <xsl:call-template>

- Предизвиква изпълнение на шаблон
- Атрибут name дефинира името на извиквания шаблон

- Елемент <xsl:with-param>

- Подаване стойност към <xsl:param> елемент
- Стойността на параметъра се определя от атрибута select

<td>

```
<xsl:call-template name="iso8601DateToDisplayDate">
```

```
  <xsl:with-param name="iso8601Date" select="@bornDate" />
```

```
</xsl:call-template>
```

</td>

<td>

```
<xsl:call-template name="iso8601DateToDisplayDate">
```

```
  <xsl:with-param name="iso8601Date" select="@diedDate" />
```

```
</xsl:call-template>
```

</td>

Шаблони с параметър

- Елементът `param` е специална променлива:

- `<xsl:param name="prefix">default</xsl:param>`

- `<xsl:with-param name="prefix">new</xsl:with-param>`

- Елементът `call-template` предава **НОВИТЕ `param` СТОЙНОСТИ** КЪМ шаблона

- ```
<xsl:template match="name">
 <xsl:call-template name="salutation">
 <xsl:with-param name="greet">Hello </xsl:with-param>
 </xsl:call-template>
</xsl:template>
```

```
<xsl:template name="salutation">
 <xsl:param name="greet">Dear </xsl:param>
 <xsl:value-of select="$greet"/>
 <xsl:apply-templates/>
</xsl:template>
```

Call to  
this  
template

The  
template

XML

XSLT

# Инициализиране на стойности за атрибути от променлива

```
<tr style="{ $rowCSS}" >
```

```
<xsl:value-of select="$rowCSS" />
```

Голяма скоба се поставя, ако атрибутът принципно не очаква като стойност XPath израз, а се налага такава да бъде изчислена

## Attribute Value Templates (AVT)

**Curly braces** are used in attribute templates to tell the **XSLT** processor to evaluate what inside each of them as an expression, rather than as normal text. In the output tree, the **curly braces** and expression are replaced with a resulting string.

# Шаблони по подразбиране (Default Templates)

- Съществуват вградени (built-in) XSLT templates:

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

```
<?xml-stylesheet type="text/xsl" href="example6.3.2.3.xsl"?>
```

```
<persons >
```

```
 <person id="123456" age="53">
```

```
 <name title="Bai">
```

```
 <first>Ganyo</first>
```

```
 <last>Balkanski</last>
```

```
 </name>
```

```
 <profession>Trader, Image Maker, Politician</profession>
```

```
 <descr>
```

Who didn't know about Bai Ganyo, who has not heard of him? ...

```
</descr>
```

```
</person>
```

```
<person id="345" age="29">
```

```
 <name title="Mr.">
```

```
 <first>Balkan</first>
```

```
 <last>Ganyov</last>
```

```
 </name>
```

```
 <profession>Facilitator</profession>
```

```
 <descr>
```

Facilitator of people who are in the middle of nowhere...

```
</descr>
```

```
</person>
```

```
</persons>
```

Стиловият документ обаче не съдържа нито една инструкция за трансформация:

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

```
<xsl:stylesheet version="1.0"
```

```
 xmlns:xsl="http://www.w3.org/1999/XSL
 /Transform">
```

```
</xsl:stylesheet>
```

# Резултат



- Представяне на документ в браузер с използване на празен XSLT документ

# Шаблони по подразбиране (Default Templates)

- Съществуват вградени (built-in) XSLT templates.
- XSLT процесорът ги ползва, ако не намери подходящ шаблон.
- Напр., ако няма шаблон за корена на документа, XSLT предоставя такъв по подразбиране, който пък прилага всички съществуващи шаблони:

```
<xsl:template match="*/">
```

```
<xsl:apply-templates/>
```

```
</xsl:template>
```

- За всеки елемент в документа (вкл. и за корена) се извиква `<xsl:apply-templates>`

# Други шаблони по подразбиране

- Вграден шаблон за текстови елементи и атрибути:  

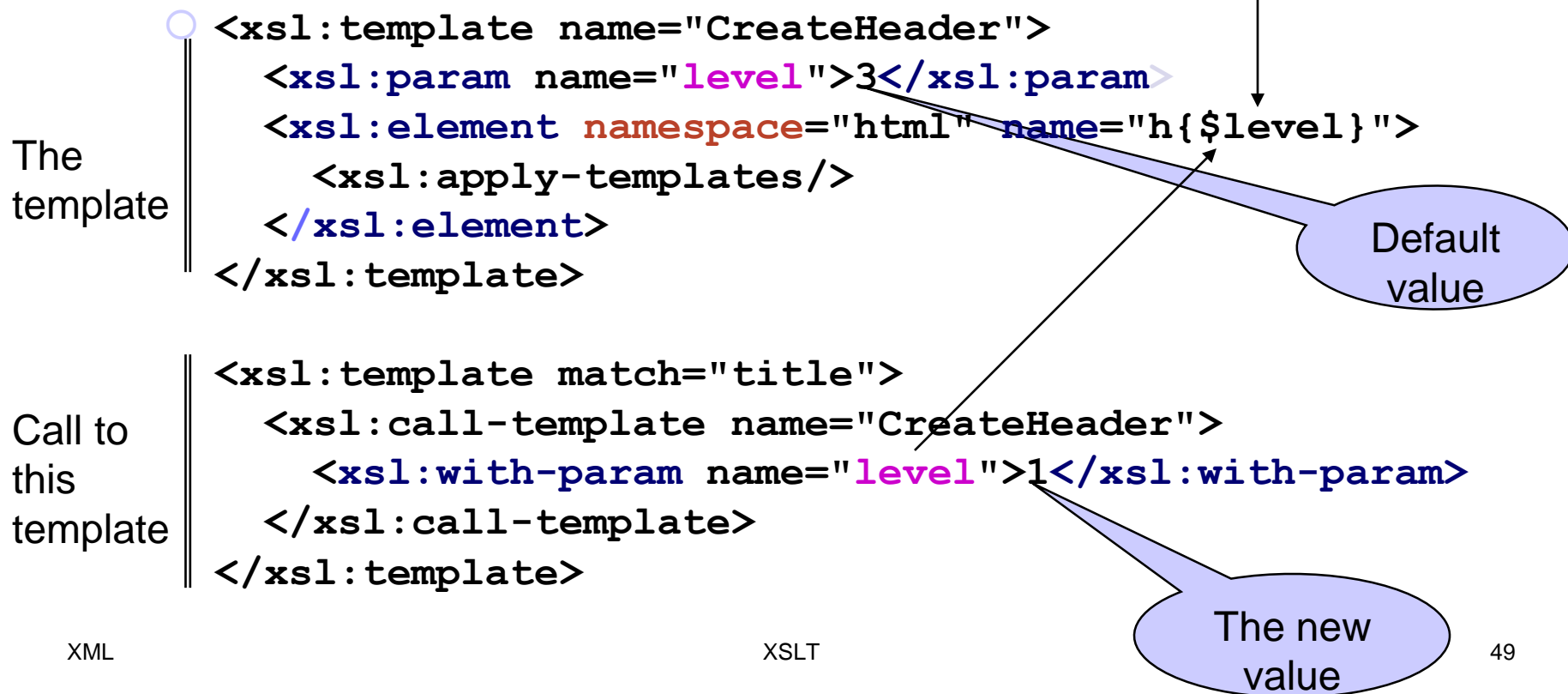
```
<xsl:template match="text()|@"*>
 <xsl:value-of select="."/>
</xsl:template>
```
- Този шаблон просто добавя стойността на текстов възел или атрибут към резултата.
- Затова получаваме резултат като този:





# Създаване на елементи – елемент `<xsl:element>`

- Елемент се създава чрез тага 'Element'
- Удобно е да се ползва с променливи



# Динамично създаване на елементи

- Пример: имената на елементите в резултатното дърво ще са съдържанието на елементите в изходящото дърво:

```
<xsl:template match="name">
 <xsl:element name="{ . }">
 Nice person!
 </xsl:element>
</xsl:template>
```

- Напр. за `<name>Milena</name>` -> резултатът е:

```
<Milena>Nice person!</Milena>
```

# Пример: трансформиране на атрибути до елементи!

## Начален XML документ:

```
<?xml version="1.0"?>
<people>
 <name first="John" middle="Fitzgerald Johansen" last="Doe"/>
 <name first="Franklin" middle="D." last="Roosevelt"/>
 <name first="Alfred" middle="E." last="Neuman"/>
 <name first="John" middle="Q." last="Public"/>
 <name first="Jane" middle="" last="Doe"/>
</people>
```

# XSL документ

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

```
<xsl:template match="/">
 <people>
 <xsl:apply-templates select="people/name"/>
 </people>
</xsl:template>
```

```
<xsl:template match="name">
 <name>
 <xsl:apply-templates select="@*"/>
 </name>
</xsl:template>
```

```
<xsl:template match="@*">
 <xsl:element name="{local-name()}"> <xsl:value-of select="."/> </xsl:element>
</xsl:template>
</xsl:stylesheet>
```

Връща името на  
текущия възел без  
namespace префикс

# Резултатен XML документ

- `<?xml version="1.0"?>`
- `<people>`
- `<name>`
- `<first>John</first>`
- `<middle>Fitzgerald Johansen</middle>`
- `<last>Doe</last>`
- `</name>`
- `.....`
- `<name>`
- `<first>Jane</first>`
- `<middle></middle>`
- `<last>Doe</last>`
- `</name>`
- `</people>`

# Копиране на елементи

- Елементът 'copy' копира контекстния възел без атрибутите и наследниците му!!!

```
<xsl:template match="h1|h2|h3|h4|h5|h6|h7">
 <xsl:copy>
 Header: <xsl:apply-templates/>
 </xsl:copy>
</xsl:template>
```

- Оригиналните атрибути не се запазват
- Атрибути създаваме чрез <xsl:attribute>

```
<xsl:template match="h1|h2|h3|h4|h5|h6|h7">
 <xsl:copy>
 <xsl:attribute name="style">purple</xsl:attribute>
 Header: </xsl:apply-templates>
 </xsl:copy>
</xsl:template>
```

XML

XSLT

`<value-of select="@style"/>`

# *attribute-set* елемент

- Използва се за създаване на групи от атрибути за по-нататъшна употреба (чрез *use-attribute-sets*)

```
<xsl:attribute-set name="class-and-color">
 <xsl:attribute name="class">standard</xsl:attribute>
 <xsl:attribute name="color">red</xsl:attribute>
</xsl:attribute-set>
```

```
<xsl:template match="h1|h2|h3|h4|h5|h6|h7">
 <xsl:copy use-attribute-sets="class-and-color">
 Header: <xsl:apply-templates/>
 </xsl:copy>
</xsl:template>
```

# ***copy-of*** елемент

Копира фрагменти от входното XML дърво без загуба на атрибути и съдържани под-елементи (*deep copy*):

```
<xsl:template match="body">
 <body>
 <xsl:copy-of select="//h1 | //h2" />
 <xsl:apply-templates/>
 </body>
</xsl:template>
```



# xsl:for-each (loop in XSLT)

- Елементът `<xsl:for-each>` селектира всеки един XML елемент от определено множество възли (node-set):

```
<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>
```

# УСЛОВИЯ - *if*

- Елементът <xsl:if>

- прилага се само ако условието му е истина
- не предоставя else клауза
- атрибут test получава стойност XPath израз (True или False)

```
<xsl:template match="para">
```

```
<html:p>
```

```
<xsl:if test="position() = 1">
```

```
<xsl:attribute name="style">color: red</xsl:attribute>
```

```
</xsl:if>
```

```
<xsl:if test="position() > 1">
```

```
<xsl:attribute name="style">color: blue</xsl:attribute>
```

```
</xsl:if>
```

```
<xsl:apply-templates/>
```

```
</html:p>
```

```
</xsl:template>
```

**Source:** [http://www.w3schools.com/Xsl/tryxslt.asp?xmlfile=catalog&xsltfile=tryxslt\\_if](http://www.w3schools.com/Xsl/tryxslt.asp?xmlfile=catalog&xsltfile=tryxslt_if)

## Пример: избор на title и artist само АКО цената на CD е по-висока от 10

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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">
 <xsl:if test="price > 10">
 <tr>
 <td><xsl:value-of select="title"/></td>
 <td><xsl:value-of select="artist"/></td>
 </tr>
 </xsl:if>
 </xsl:for-each>
 </table> </body> </html>
</xsl:template>
</xsl:stylesheet>
```

*Други примери:*

[http://www.w3schools.com/xsl/el\\_if.asp](http://www.w3schools.com/xsl/el_if.asp)

**No. 1 for Windows**  
**Web Hosting**  
 2008 - 2009 - 2010

**TRY IT FOR FREE**  
 Start my free trial  
**fonts.com** web  
 webfonts.fonts.com

XML Code: Edit and Click Me >> XSLT Code:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by XMLSpy® -->
<catalog>
 <cd>
 <title>Empire Burlesque</title>
 <artist>Bob Dylan</artist>
 <country>USA</country>
 <company>Columbia</company>
 <price>10.90</price>
 <year>1985</year>
 </cd>
 <cd>
 <title>Hide your heart</title>
 <artist>Bonnie Tyler</artist>
 <country>UK</country>
 <company>CBS Records</company>
 <price>9.90</price>
 <year>1988</year>
 </cd>
 <cd>
 <title>Greatest Hits</title>
 <artist>Dolly Parton</artist>
 <country>USA</country>
 <company>RCA</company>
 <price>9.90</price>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by XMLSpy® -->
<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">
 <xsl:if test="price>10">
 <tr>
 <td><xsl:value-of select="title"/></td>
 <td><xsl:value-of select="artist"/></td>
 </tr>
 </xsl:if>
 </xsl:for-each>
 </table>
 </body>
 </html>
</xsl:template>
```

Your Result:

## My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Still got the blues	Gary Moore

XSLT

# УСЛОВИЯ – *xsl:choose* (1/2)

- `<xsl:choose>` се ползва заедно с `<xsl:when>` и `<xsl:otherwise>`
- задава многократни условни тестове

```
<xsl:template match="para">
 <html:p>
 <xsl:choose>
 <xsl:when test="position() = 1">
 <xsl:attribute name="style">color: red</xsl:attribute>
 </xsl:when>
 <xsl:otherwise test="position() > 1">
 <xsl:attribute name="style">color: blue</xsl:attribute>
 </xsl:otherwise>
 <xsl:apply-templates/>
 </xsl:choose>
</html:p>
</xsl:template>
```

← *опционално*

# Условия – *xsl:choose* (2/2)

- `<xsl:choose>`
- `<xsl:when test="salary[number(.) > 2000]">`  
A big number for BG
- `</xsl:when>`
- `<xsl:when test="salary[number(.) > 1000]">`  
A medium number for BG
- `</xsl:when>`
- `<xsl:otherwise>`A small number
- `</xsl:otherwise>`
- `</xsl:choose>`

# Елементът <xsl:sort> 1/2

- Използва се като дъщерен елемент на <xsl:apply-templates> или <xsl:for-each> елементите
- Атрибутът select
  - XPath израз рефериращ възел(и) за сортиране
- Атрибутът data-type
  - Начин на тълкуване на стойностите за възлите (обикновено се използва текст или число)
- Атрибутът order
  - Ред на сортиране (възходящ или низходящ)

# Елементът <xsl:sort> 2/2

- Пример: сортиране по дата

- Необходимост от преобразуване на датата в число с функцията `translate()`

```
translate('The first of the few ', 'fiw', 'wot')
```

```
 The worst of the wet.
```

```
translate(@bornDate, '-' , '')
```

- Използване на елемента **<xsl:sort>**

```
<xsl:apply-templates select="People/Person">
```

```
 <xsl:sort select="translate(@bornDate, '-' , '')"
 data-type="number"/>
```

```
</xsl:apply-templates>
```



# Два механизма за комбиниране на набори от стилове

- механизъм за включване, който позволява да бъдат комбинирани стилове, без да се променя семантиката на стиловите, които се съчетават - посредством елемента
  - **<xsl:include href=uri-reference />**
- механизъм за внасяне, който позволява на стилове да се отменят един друг - с използване на елемента
  - **<xsl:import href=uri-reference />**

# Елементът <xsl:include> 1/2

- Извършва вмъкване на XSLT документ в друг XSLT документ
- Осигурява изграждане на модули
- Предимства
  - Преизползваемост на кода
  - Спестяване на усилия при необходимост от промяна

# Елементът <xsl:include> 2/2

```
<xsl:include href="DateTemplates.xslt" />
```

- Атрибут href
  - Задава местоположението и името на XSLT документа
- Начин на обработка от XSLT процесора
  - Премахва се външния <xsl:stylesheet> елемент
  - Изгражда в паметта документ, който включва основния документ и добавените документи с елемента <xsl:include>

# Елементът <xsl:import>

- Осигурява функционалност подобна на <xsl:include>
- Ако импортираният шаблон е в конфликт с някой от шаблоните в основния XSLT документ, то той получава по-нисък приоритет
- Приложим е при многократна обработка на възли
  - Пример: изграждане на съдържание или резюме
  - Използват се няколко <xsl:template> елемента с атрибут mode

# XSLT 2.0

The header area features the text 'XSLT 2.0' in a large, black, sans-serif font. To the right of the text are five circles arranged in a horizontal row. The first circle is solid light purple. The second circle is a white outline. The third circle is solid light purple. The fourth circle is a white outline. The fifth circle is solid light purple.

Строго типизиране

Потребителски дефинирани функции

Множество изходни документи от една трансформация

Едновременно обработка на множество документи: функция `collection()`

Поддръжка на групиране

Обработка на файлове, които не са в XML формат, напрмер CSV

По-добра обработка на текст: регулярни изрази

Поддръжка на XPath 2.0

# Типове данни в XSLT 2.0

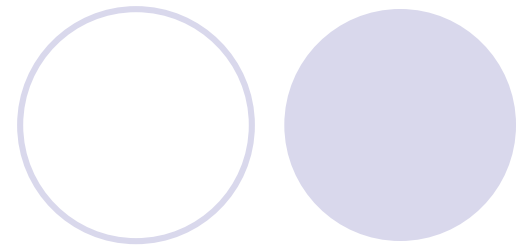
- XSLT 1.0

- Текстови, числени и булеви стойности

- XSLT 2.0

- Поддръжка множество типове, базирани на XML Schema: integer, double, decimal, day, month, year и др.
  - Възможност за импортиране на типове от други XML схеми
  - Поддръжка на 2 типа процесори
    - basic
    - schema-aware
  - Генериране на грешки от процесора при несъответствие на типовете
  - <http://www.w3.org/TR/xpath-functions/#datatypes>

# Елемент <xsl:function> - дефиниране на функция



```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:myData="http://wrox.com/namespaces/embeddedData"
 xmlns:myFunc="http://wrox.com/namespaces/functions/datetime"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
 exclude-result-prefixes="myFunc">
<xsl:variable name="thisDocument" select="document('')" />
...
<xsl:function name="myFunc:iso8601DateToDisplayDate" as="xs:string">
 <xsl:param name="iso8601Date" as="xs:date" />
 <xsl:variable name="yearPart" select="year-from-date($iso8601Date)"
as="xs:integer" />
 <xsl:variable name="monthPart" select="month-from-date($iso8601Date)"
as="xs:integer" />
 <xsl:variable name="monthName"
select="$thisDocument/xsl:stylesheet/myData:Months/Month[@index =
number($monthPart)]"
/>
 <xsl:variable name="datePart" select="day-from-date($iso8601Date)"
as="xs:integer" />
 <xsl:value-of select="concat($datePart, ' ', $monthName, ' ', $yearPart)" />
</xsl:function>
</xsl:stylesheet>
```

# Елемент <xsl:function> - извикване на функция

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:myFunc="http://wrox.com/namespaces/functions/datetime">
```

...

```
<tr style="{ $rowCSS}">
 <td style="{ $nameCSS}">
<xsl:value-of select="Name" /></td>
 <td><xsl:value-of select="myFunc:iso8601DateToDisplayDate(@bornDate)" />
</td>
 <td><xsl:value-of select="myFunc:iso8601DateToDisplayDate(@diedDate)" />
</td>
 <td> <xsl:value-of select="Description" /></td>
</tr>
```

```
.NET transform -s:people.xml -xsl:peopleToHtml-
UsingFunctions.xslt
-o:people-usingFunctions.html
```

```
Java java net.sf.saxon.Transform -s:people.xml
-xsl:peopleToHtml-UsingFunctions.xslt -
o:people-usingFunctions.html
```



# елементът <xsl:function> 1/2

- Дефиниция

```
<xsl:function name="myFunc:iso8601DateToDisplayDate" as="xs:string">
 <xsl:param name="iso8601Date" as="xs:date" />
 <!-- rest of function -->
</xsl:function>
```

- Атрибутът name

- Включва префикс, рефериращ към пространство с имена

- Атрибутът as

- Специфицира типа на резултата

- Елементът <xsl:param>

- Дефинира параметър на функцията от определен тип, зададен с атрибут **as**

# елементът <xsl:function> 2/2

- Елементът <xsl:variable>

- Дефинира променлива във функцията от определен тип, зададен с атрибут **as**

- Функцията year-from-date()

```
<xsl:variable name="yearPart"
 select="year-from-date($iso8601Date)" as="xs:integer" />
```

- Функцията format-date()

```
<xsl:value-of select="format-date(@bornDate, '[D1] [MNn] [Y]')"/>
```

# Използване на XSLT при клиента

- Ако браузърът поддържа XSLT, то тази технология може да се ползва за трансформиране на документа до XHTML в самия браузър.
- По-гъвкаво решение е да се ползва за трансформацията и JavaScript, с цел:
  - Тестване на специфични за браузъра черти
  - Използване на различни style sheets според браузъра и нуждите на потребителя

# Пример

(<http://www.w3schools.com/xsl>)

```
...
function displayResult() {
 xml=loadXMLDoc("cdcatalog.xml");
 xsl=loadXMLDoc("cdcatalog.xsl");
 // code for IE
 if (window.ActiveXObject) {
 ex=xml.transformNode(xsl);
 document.getElementById("example").innerHTML=ex;
 } // code for Mozilla, Firefox, Opera, etc.
 else if (document.implementation &&
document.implementation.createDocument) {
 xsltProcessor=new XSLTProcessor();
 xsltProcessor.importStylesheet(xsl);
 resultDocument = xsltProcessor.transformToFragment(xml,document);
 document.getElementById("example").appendChild(resultDocument);
 }
}
```

# Сървърна XSLT

- Не всички браузъри поддържат XSLT =>
- Разумно решение е да трансформиране XML документа до XHTML на сървъра
- Отново можем да отразим при трансформацията специфични черти на даден браузър
- Бързина и универсалност

# DOM спрямо XSL



- За по-сложно реструктуриране и сортиране на документа, използваме DOM
- При DOM ние парсваме XML документа, после изпълняваме код за манипулиране на DOM дървото (напр. на Java) според дадени изисквания.
- Този код има пълен достъп до методите на DOM и на дървото, така че не сме ограничени от лимитираните възможности на XSL

# Алтернатива на XSLT: W3C XQuery (a query language for XML)

- XQuery се ползва за XML публикуване на Web съобщения, динамични Уеб сайтове и др. приложения.
- Имплементация: DataDirect Xquery (<https://www.progress.com/xquery/xml>)
- Повечето от функционалността на XQuery, като напр. аритметични оператори, сравнения, работа с функции и др. са познати на програмистите
- Три отличителни особености на XQuery:
  - **Path изрази** - позволяват локиране на части от XML документ.
  - **XML конструктори** - за създаване на произволен XML документ.
  - **FLWOR** (произнасяно "flower") **изрази**: For, Let, Where, Order By, and Return – разрешават комбиниране на данни за създаване на нови XML структури. Подобни са на SQL Select statements и техните From и Where клаузи, но са адаптирани за XML обработка.

# За повече информация – вижте спецификациите

Part	Date	Status	URL
<b>XSL- Formatting Objects ver. 2.0</b>	<b>04.02.2010</b>	<b>Version 2.0 W3C Working Draft</b>	<b><a href="http://www.w3.org/TR/xslfo20/">http://www.w3.org/TR/xslfo20/</a></b>
<b>XSL Transformations (XSLT) ver. 2.1</b>	<b>11.05.2010</b>	<b>Version 2.1 W3C Working Draft</b>	<b><a href="http://www.w3.org/TR/xslt-21/">http://www.w3.org/TR/xslt-21/</a></b>
<b>XML Path Language (XPath) ver. 2.0</b>	<b>23.01.2007</b>	<b>Version 2.0 W3C Working Draft</b>	<b><a href="http://www.w3.org/TR/xpath20/">http://www.w3.org/TR/xpath20/</a></b>