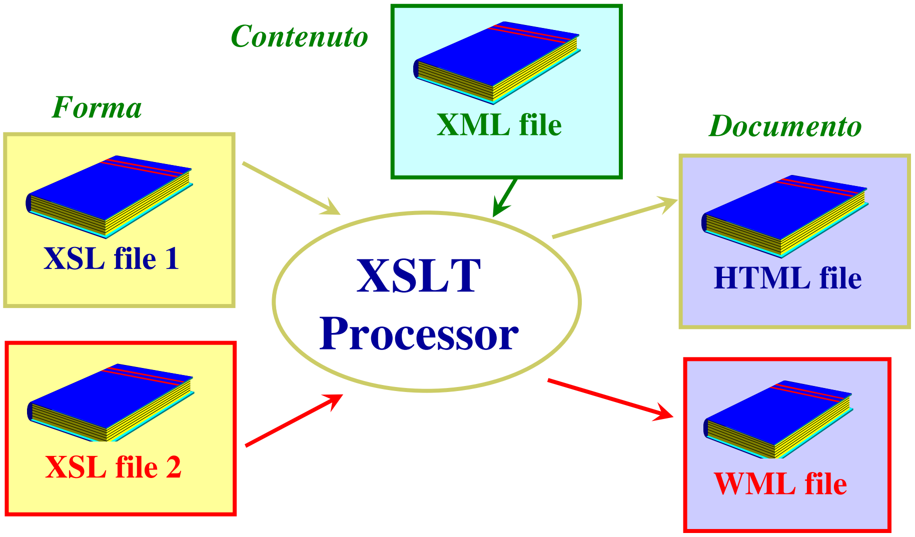
Introduction to XSL

XSL-BASIC ELEMENTS

Transforming XML



Contenuto

XML file

Forma

XSL file 1

XSL file 2

XSLT Processor

Documento

HTML file

WML file

HANDS ON! - Esempio1 XML

```
<?xml version="1.0"?>
<?xml-stylesheet href="hello.xsl" type="text/xsl"?>

<!-- Here is a sample XML file -->
<page>
  <title>Test Page</title>
  <content>
    <paragraph>What you see is what you get!</paragraph>
  </content>
</page>
```

HANDS ON! - Esempio1 XSL a

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="page">
    <html>
      <head>
        <title>
          <xsl:value-of select="title"/>
        </title>
      </head>
      <body bgcolor="#ffffff">
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
```

HANDS ON! - Esempio1 XSL b

```
    <xsl:template match="paragraph">
        <p align="center">
            <i>
                <xsl:apply-templates/>
            </i>
        </p>
    </xsl:template>
</xsl:stylesheet>
```

HANDS ON! - Esempio1 Xalan

Let us use the Apache XSLT processor: Xalan.

1) Get Xalan from xml.apache.org/xalan/index.html

2)Set CLASSPATH=%CLASSPATH%;.../xalan.jar;   .../xerces.jar

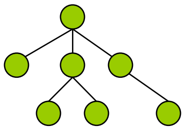
3) java **org.apache.xalan.xslt.Process**
–IN testPage.xml –XSL testPage.xsl –O out.html

HANDS ON! - Esempio1 Output HTML

```
<html>
  <head>
    <title>
      Test Page
    </title>
  </head>
  <body bgcolor="#ffffff">
    <p align="center">
      <i>
        What you see is what you get!
      </i>
    </p>
  </body>
</html>
```

The process

- The process starts by traversing the document tree, attempting to find a single matching rule for each visited node.
- Once the rule is found, the body of the rule is istantiated
- Further processing is specified with the *<xsl:apply-templates>*. The nodes to process are specified in the *match* attribute. If the attribute is omitted, it continues with the next element that it has a matching template.



Implicit rules

```
<template match=“/|*”>
  <apply-templates/>
</template>

<template match=“text()”>
  <value-of select=“.”/>
</template>
```

Selective processing

```
<template match=“group”>
  <apply-templates select=“name”>
</template>

<template match=“intro”>
  <apply-templates select=“//chapter/title”>
</template>
```

Selective processing - example

```
<?xml version="1.0"?>
<?xml-stylesheet href="IgnoraParte4.xsl" type="text/xsl" ?>
<ROOT>
<SECRET>
SEZIONE RISERVATA:
  <TAG1>Testo Privato</TAG1>
</SECRET>
<PUBLIC>
SEZIONE PUBBLICA
 <TAG1>Testo Pubblico</TAG1>
</PUBLIC>
</ROOT>
```

Selective processing - example

```
<xsl:stylesheet    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
>
  <xsl:template match="SECRET">Esiste una parte privata</xsl:template>
  <xsl:template match="PUBLIC">Esiste una parte pubblica</xsl:template>
  <xsl:template match="PUBLIC">La parte pubblica contiene:<xsl:apply-
    templates/></xsl:template>
</xsl:stylesheet>
```

OUTPUT

```
<?xml version="1.0" encoding="UTF-8"?>

Esiste una parte privata
La parte pubblica contiene:
SEZIONE PUBBLICA
Testo Pubblico
```

Pattern Matching - nodes

/ matches the root node

A matches any <A> element

* matches any element

A|B matches any <A> or <B> element

A/B matches any <B> element within a <A> element

A//B matches any <B> element with a <A> ancestor

text() matches any text node

Pattern Matching

id(“pippo”) matches the element with unique ID pippo

A[1] matches any <A> element that is the first <A> child of its parent

A[last()=1] matches any <A> element that is the last <A> child of its parent

B/A[position() mod 2 = 1] matches any <A> element that is an odd-numbered <A> child of its B parent

Pattern Matching - attributes

@A matches any A attribute

@* matches any attribute

B[@A=“v”]//C matches any <C> element that has a <B> ancestor with a A attribute with v value

processing-instruction()
node()

Imports, priorities and spaces

IMPORT

<import href=“…”>

PRIORITIES

<template match=“…” priority=“2” > (default 1)

When priorities are equal, the last definition wins

STRIPPING SPACES

```
<xsl:stylesheet     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
<xsl:strip-space elements="*"/>
...
</xsl:stylesheet>
```

Variables, templates and parameters

```
<variable name=“colore”>rosso</variable>
…
Il colore e’: <xsl:value-of select=“$colore”>.

<template name=“header”>
Sequence of text and tags
</template>
…
<call-template name=“header”/>

<template name=“header”><param name=“P”>default</param>
Sequence of text and tags, including <value-of select=“$P”/>
</template>
…
<call-template name=“header”>
<with-param name=“P”>3</with-param></call-template>
```

Once a value has been assigned to a variable, it cannot be changed

conditions

```
< xsl: if test”position() mod 2 =0”>
    <B><apply_templates/></B>
</ xsl: if>

<xsl:choose>
<xsl: when test”position() mod 2 =0”>
    <B><apply_templates/></B>
</xsl: when>
<xsl: otherwise>
    <I><apply_templates/></I>
</xsl: otherwise >
</xsl: choose >
```

for-each

```
<xsl:for-each select="expression">
    some rule
</xsl:for-each>
```

Sorting

```
<list>
    <item sortcode="C"> Pluto</item>
    <item sortcode="A"> Topolino </item>
    <item sortcode="B">Pippo</item>
</list>

<template match="list">
 <apply-templates><sort/></apply-templates>
</template>

<template match="list">
 <apply-templates>
 <sort select="@sortcode" order=descending/>
</apply-templates>
</template>
```

Numbering

```
<list>
    <item sortcode=“C”> Pluto</item>
    <item sortcode=“A”> Topolino </item>
    <item sortcode=“B”>Pippo</item>
</list>

<template match=“item”>
 <apply-templates><number format=“A”/></apply-templates>
</template>
```

Numbering

```
<chapter>
  <section>
    <title> First section of chapter 1</title>
    …

<template match=“section/title”>
 <number level=“multi” format=“1.A”/ count=chapter|section/>
</template>
```