Exercises with XSLT

Movies xml

<?xsl-stylesheet ->

Links to xslt

Movies xml

xslt

movies xml link dummy with xslt

xslt uses document(movies.xml)

Different ways

Before we start doing XSLT

Movies xml .xml <?xsl-stylesheet>

XSLT .xsl

Movies xml

Xslt

dummy.xml is one I want to open

links to xslt

document(“movies.xml”)

open dummy one gets xslt one using movies

First one link movies xml to xslt and then look at second one.

Want to have an output that is XML and we want each actor ,every actor and actors are inside the movies.

<Actors>  
<Actor Name=”” YOB=”” Country=”” >

<Actor ……./>

</Actors>

1. **xsl:copy-of**: This element is used to create a copy of the nodes selected by the **select** attribute. The copied nodes are inserted into the output document.
2. **select="Movies/Movie/Actor[not(@Name=following::Actor/@Name)]"**: This is an XPath expression used to select which nodes should be copied. It specifies the selection of **Actor** elements under **Movie** elements, which are themselves under **Movies**. The nodes are selected based on a condition:

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

<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1">

<xsl:output method="xml"/>

<xsl:template match="/">

<Actors>

<xsl:copy-of select="Movies/Movie/Actor[not(@Name=following::Actor/@Name]">

</Actors>

</xsl:template>

</xsl:transform>

Start from the top,start building the content

* **[not(@Name=following::Actor/@Name)]**: This is a predicate (a condition inside square brackets) that filters which **Actor** nodes to select. It uses the **not()** function to include only those **Actor** nodes where there are no following **Actor** nodes with the same **@Name** attribute within the entire document.

**Helps to remove duplicates**

To sort them it requires a loop.

Couple of options when we are doing loops.

Want to do a for-each loop.

Looping through actor element nodes.

Relative to thing in the loop.

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

<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1">

<xsl:output method="xml"/>

<xsl:template match="/">

<Actors>

<xsl:for-each select="Movies/Movie/Actor[not(@Name=following::Actor/@Name)]">

<xsl:sort select="@Name">

</xsl:for-each>

</Actors>

</xsl:template>

</xsl:transform>

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

<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1">

<xsl:output method="xml"/>

<xsl:template match="/">

<Actors>

<xsl:for-each select="Movies/Movie/Actor[not(@Name=following::Actor/@Name)]">

<xsl:sort select="@Name"/>

<xsl:copy-of select="current()"/>

</xsl:for-each>

</Actors>

</xsl:template>

</xsl:transform>

Copy-of

Don’t want all the attributes,want to keep only the names.

In that case want to create a new actor.

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

<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1">

<xsl:output method="xml"/>

<xsl:template match="/">

<Actors>

<xsl:for-each select="Movies/Movie/Actor[not(@Name=following::Actor/@Name)]">

<xsl:sort select="@Name"/>

<Actor Name=”{@Name}”></Actor>

</xsl:for-each>

</Actors>

</xsl:template>

</xsl:transform>

Want to put name not as an attribute but as an actual content,

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

<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1">

<xsl:output method="xml"/>

<xsl:template match="/">

<Actors>

<xsl:for-each select="Movies/Movie/Actor[not(@Name=following::Actor/@Name)]">

<xsl:sort select="@Name"/>

<Actor Name="{@Name}"></Actor>

</xsl:for-each>

</Actors>

</xsl:template>

</xsl:transform>

For {} curly braces need to have XSLT 3

If only have XSLT 1 need to use some

<Actor> <xsl:value-of select=”@Name”></Actor>

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

<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1">

<xsl:output method="xml"/>

<xsl:template match="/">

<Actors>

<xsl:for-each select="Movies/Movie/Actor[not(@Name=following::Actor/@Name)]">

<xsl:sort select="@Name"/>

<xsl:value-of select="@Country"/>

</xsl:for-each>

</Actors>

</xsl:template>

</xsl:transform>

How about not using hard coded elements Actors and Actor.

Maybe don’t want to have actors created this way.

Can use instructions that are node creation instructions.

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

<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1">

<xsl:output method="xml"/>

<xsl:template match="/">

<xsl:element name="Actors">

<xsl:for-each select="Movies/Movie/Actor[not(@Name=following::Actor/@Name)]">

<xsl:sort select="@Name"/>

<xsl:element name="Actor">

<xsl:attribute name="TheName">

<xsl:value-of select="@Name"></xsl:value-of>

</xsl:attribute>

</xsl:element>

</xsl:for-each>

</xsl:element>

</xsl:template>

</xsl:transform>

XML document with every Actor as an element.

HTML document with all actors as a HTML table.

Actors

One row with headings

Something like Name,YOB and Country.

Name YOB Country

Sylvester Stallone

Woody Allen

2nd way of linking – dummy file

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

<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1">

<xsl:output method="html"/>

<xsl:variable name="d" select="document('movies.xml')"/>

<xsl:template match="/">

<html>

<head>

<title>

Actors</title>

</head>

<body>

<h1>Actors</h1>

</body>

</html>

</xsl:template>

</xsl:transform>

Open movies.xml and put it in a variable d.

Then it will start doing content of this template,which is right now hard coded.

It will create Actors,the heading before the table.

Don’t have the table yet in here.

Xt2.xsl

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

<?xml-stylesheet type="text/xsl" href="xt2.xsl"?>

<Dummy/>

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

<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1">

<xsl:output method="html"/>

<xsl:variable name="d" select="document('movies.xml')"/>

<xsl:template match="/">

<html>

<head>

<title>

Actors</title>

</head>

<body>

<h1>Actors</h1>

<table border="2" width="60%">

<tr>

<th>Name</th>

<th>YoB</th>

<th>Country</th>

</tr>

</body>

</html>

</xsl:template>

</xsl:transform>

<xsl:apply-templates select=” Movies/Movie/Actor[not(@Name=following::Actor/@Name)]">”>

<xsl:sort select=”@Name”>

</xsl:apply-templates>

</table> </body> </html>

</xsl:template></xsl:transform>

3.Movies per director

Go to relevant web pages under XSLT, see 2 links

xsltransform.net

see what engine it uses internally.

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

<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="html"/>

<xsl:variable name="doc" select="document('movies.xml')" />

<xsl:template match="/">

<html>

<head><title>Movies per director (XT3)</title></head>

<body>

<h1>Movies per director</h1>

<ul>

<xsl:apply-templates select="$doc//Director[not(@Name = preceding::Director/@Name)]">

<xsl:sort select="@Name"/>

</xsl:apply-templates>

</ul>

</body>

</html>

</xsl:template>

<xsl:template match="Director">

<li>

<xsl:value-of select="@Name"/>

<ol>

<xsl:apply-templates select="$doc//Movie[Director/@Name = current()/@Name]">

<xsl:sort select="@Year"/>

</xsl:apply-templates>

</ol>

</li>

</xsl:template>

<xsl:template match="Movie">

<li>

<xsl:value-of select="@Title"/> (<xsl:value-of select="@Year"/>)

</li>

</xsl:template>

</xsl:transform>

The XSLT code you've shared transforms an XML document containing information about movies and directors into HTML format, listing all movies grouped by director in a sorted order. Let's break down how it works:

**XSLT Code Explanation:**

1. **XSLT Stylesheet Declaration:**

xmlCopy code

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

This sets up the XSLT processor with the correct version and namespace.

1. **Output Method:**

xmlCopy code

<xsl:output method="html"/>

Specifies that the output should be HTML.

1. **Variable Declaration:**

xmlCopy code

<xsl:variable name="doc" select="document('movies.xml')"/>

Defines a variable **doc** that loads an external XML document named **movies.xml**, which presumably contains the data about movies and directors.

1. **Root Template:**

xmlCopy code

<xsl:template match="/"> <html> <head><title>Movies per director (XT3)</title></head> <body> <h1>Movies per director</h1> <ul> <xsl:apply-templates select="$doc//Director[not(@Name = preceding::Director/@Name)]"> <xsl:sort select="@Name"/> </xsl:apply-templates> </ul> </body> </html> </xsl:template>

* + Matches the root of the input XML document.
  + Constructs the basic HTML structure with a title and a heading.
  + Initiates an **apply-templates** instruction for each **Director** element in the external document, ensuring each director's name appears only once using a uniqueness check (**not(@Name = preceding::Director/@Name)**).
  + Sorts the directors by their **@Name** attribute.

1. **Template for Director:**

xmlCopy code

<xsl:template match="Director"> <li> <xsl:value-of select="@Name"/> <ol> <xsl:apply-templates select="$doc//Movie[Director/@Name = current()/@Name]"> <xsl:sort select="@Year"/> </xsl:apply-templates> </ol> </li> </xsl:template>

* + Matches each **Director** element.
  + Creates a list item for each director, displaying the director's name.
  + Generates an ordered list of movies for this director, applying templates to **Movie** elements where the **Director**'s **@Name** matches the current director.
  + Sorts movies by their **@Year** attribute.

1. **Template for Movie:**

xmlCopy code

<xsl:template match="Movie"> <li> <xsl:value-of select="@Title"/> (<xsl:value-of select="@Year"/>) </li> </xsl:template>

* + Matches each **Movie** element.
  + Creates a list item for each movie, displaying the movie's title and year.

**How It Works to Group Movies by Director:**

* The XSLT pulls data from an external XML file (**movies.xml**) using the declared variable **doc**.
* It generates an HTML list of directors, each with a sublist of their movies.
* Movies are sorted by year, and directors are sorted by name, ensuring an organized display.Top of Form

List item for each director which consists of the movies.

Exercise XT4 Desired result: An HTML document with all the movies produced by Tri-Star. Include information about each movie's director and actors.

This XSLT code generates an HTML document that lists all movies produced by "Tri-Star" from a provided XML file, including information about each movie's director and actors. Here's a detailed breakdown of how the code accomplishes this:

**XSLT Code Explanation:**

1. **XSLT Stylesheet Declaration:**

xmlCopy code

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

This sets up the XSLT processor with the correct version and namespace, and defines the transformation.

1. **Output Method:**

xmlCopy code

<xsl:output method="html"/>

Specifies that the output should be HTML, which structures the output accordingly (using HTML tags).

1. **Variable Declaration:**

xmlCopy code

<xsl:variable name="doc" select="document('movies.xml')"/>

Loads an external XML document named **movies.xml** into the variable **doc**. This document presumably contains the data about various movies, their production companies, directors, and actors.

1. **Root Template:**

xmlCopy code

<xsl:template match="/"> <html> <head><title>Movies by Tri-Star (XT4)</title></head> <body> <h1>Movies by Tri-Star</h1> <xsl:apply-templates select="$doc//Movie[ProductionCompany = 'Tri-Star']"/> </body> </html> </xsl:template>

* + Matches the root of the input XML document.
  + Constructs the basic HTML structure with a title and a main heading.
  + Uses the **apply-templates** instruction to process each **Movie** element within the **doc** variable where the **ProductionCompany** element equals "Tri-Star".

1. **Template for Movie:**

xmlCopy code

<xsl:template match="Movie"> <h2><xsl:value-of select="@Title"/> (<xsl:value-of select="@Year"/>)</h2> <h3>Director</h3> <p><xsl:value-of select="Director/@Name"/></p> <h3>Actors</h3> <xsl:for-each select="Actor"> <p><xsl:value-of select="@Name"/></p> </xsl:for-each> </xsl:template>

* + Matches each **Movie** element processed by the previous template.
  + For each movie, it creates an HTML structure displaying the movie's title and year within **<h2>** tags.
  + It then adds a section for the director's name under **<h3>** followed by **<p>**.
  + For the actors, it iterates over each **Actor** element within the current **Movie** element using **xsl:for-each**, displaying each actor's name in **<p>** tags.

**How It Works to List Movies by Tri-Star:**

* The XSLT selectively applies transformations only to **Movie** elements from **movies.xml** where the production company is "Tri-Star".
* For each selected movie, the script generates an HTML segment containing the movie's title, year, director's name, and a list of actors.
* This results in an organized display of Tri-Star movies, grouped under a common header in the HTML output, with detailed information for each film.

The feature in the provided XSLT code snippet that is specifically supported by XSLT 3.0 and not by earlier versions is the use of the **expand-text** attribute in the **<xsl:template>** element. Let's take a closer look at what this feature does:

**expand-text Attribute**

<xsl:template match="Movie" expand-text="yes">

* **expand-text="yes"**: This attribute enables text value templates within the template. With **expand-text** set to "yes", you can directly include curly braces **{}** to evaluate XPath expressions within text nodes. This makes it easier to embed XPath expressions directly inside text content without needing to use **<xsl:value-of>** or **<xsl:sequence>**. It simplifies the syntax and improves readability, especially when you need to concatenate strings or include dynamic content within text nodes.

Here is how text value templates are used in the template:

xmlCopy code

<h2>{@Title} ({@Year})</h2> <p>{Director/@Name}</p> <p>{@Name}</p>

Each of these instances uses curly braces **{}** to embed XPath expressions directly. This feature is very handy because it reduces the verbosity typical of XSLT, where you would normally have to write something like:

xmlCopy code

<h2><xsl:value-of select="@Title"/> (<xsl:value-of select="@Year"/>)</h2> <p><xsl:value-of select="Director/@Name"/></p>

This usage of **expand-text** is only available in XSLT 3.0, introduced as part of the many enhancements to streamline and modernize the language, making it more concise and easier to write for common tasks. Earlier versions of XSLT, such as XSLT 1.0 or 2.0, do not support this feature and require more verbose constructs to achieve the same results.

Top of Form

5.An XML document with all the movies grouped by production company and year. Include information about each movie's director and the actors grouped by country.

The provided XSLT code assumes the existence of a **movies.xml** document and is structured to generate a new XML where movies are grouped first by production company and then by year, with additional structuring for actors by country:

xmlCopy code

<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:output method="xml"/> <xsl:template match="/"> <xsl:variable name="doc" select="document('movies.xml')"/> <Movies> <!-- Iterate over unique production companies --> <xsl:for-each select="$doc//ProductionCompany[not(text() = preceding::ProductionCompany/text())]"> <xsl:sort select="." order="ascending"/> <xsl:variable name="pc" select="."/> <ProductionCompany Name="{text()}"> <!-- For each production company, find movies by unique years --> <xsl:for-each select="$doc//Movie[ProductionCompany = $pc and not(@Year = preceding::Movie[ProductionCompany = $pc]/@Year)]/@Year"> <ProducedMovies Year="{.}"> <xsl:apply-templates select="$doc//Movie[ProductionCompany = $pc and @Year = current()]"/> </ProducedMovies> </xsl:for-each> </ProductionCompany> </xsl:for-each> </Movies> </xsl:template> <!-- Template for each Movie --> <xsl:template match="Movie"> <xsl:variable name="m" select="."/> <Movie> <xsl:copy-of select="@Title"/> <xsl:copy-of select="Director"/> <!-- Group actors by country --> <xsl:for-each select="Actor[not(@Country = preceding-sibling::Actor/@Country)]/@Country"> <xsl:sort select="." order="ascending"/> <Actors Country="{.}"> <xsl:apply-templates select="$m/Actor[@Country = current()]"/> </Actors> </xsl:for-each> </Movie> </xsl:template> <!-- Template for each Actor --> <xsl:template match="Actor"> <Actor> <xsl:copy-of select="@Name"/> <xsl:copy-of select="@YearOfBirth"/> </Actor> </xsl:template> </xsl:transform>

**Detailed Steps:**

1. **Root Template**: Sets up a variable to load the **movies.xml** document. Begins constructing the root **Movies** element.
2. **Grouping by Production Company**: Iterates over each unique production company. This uniqueness check is done by comparing the current **ProductionCompany** text against all preceding **ProductionCompany** text nodes.
3. **Sorting and Filtering by Year**: For each production company, it iterates over movies, grouping them by unique years. This is ensured by checking that the year of the current movie has not appeared before for the same production company.
4. **Applying Movie Template**: For each grouped movie, it applies the **Movie** template to render individual movie details.
5. **Movie Template**:
   * Copies the title attribute directly.
   * Copies the **Director** element.
   * Groups actors by their country attribute. This is done by selecting unique countries from the actors and sorting them.
6. **Actor Template**: Copies relevant attributes of actors (name and year of birth), ensuring they are rendered correctly under each country grouping.

This XSLT is designed to handle complex grouping and sorting tasks and outputs a structured XML document based on specific organizational requirements.