# *XSLT Constructors*

Martin Holmes

## XSLT Constructors

- Normally, to create a result element in your output, you just type it literally:

```
<div class="chapter"> [...] </div>
```

- However, there are some circumstances when you can't do that. For instance, the name of the element you want to create may change based on the XML input. For instance:
- If you're processing a TEI <list> element, you may need to produce either <ul> or <ol> in your output, depending on whether it's an unordered or an ordered list:
-

```
<list type="bulleted"> # <ul>
```

```
<list type="ordered"> # <ol>
```

- We can handle this with the XSLT *element constructor*.

## The element constructor

```
<xsl:element name="h2">
  [...contents of the h2 element...]
</xsl:element>
```

## An example element constructor

```
<xsl:element name="{if (@type='ordered') then 'ol' else 'ul'}">
  [...]
</xsl:element>
```

- The XPath logic is contained in {curly braces}.
- Be careful with nested quotes!

## The attribute constructor

Just as you can create an element with a constructor, you can also create an attribute:

```
<xsl:attribute name="class">
  [... value of the class attribute ...]
</xsl:attribute>
```

You might do this if you need to generate the attribute value dynamically based on the content.

Like `<xsl:variable>`, the value can be given as the value of a `@select` attribute, or as the content.

## An example attribute constructor

```
<xsl:template match="div | p | ab">
  <div>
    <xsl:attribute name="class" select="local-name()"/>
    <xsl:apply-templates/>
  </div>
</xsl:template>
```

## XSL constructors: Task 1

Write a constructor for an element called `<div>` with an attribute called "class", whose value is "chapter".

## XSL constructors: Task 1 answer

A constructor for an element `<div>` with a `@class` attribute whose value is `"chapter"`:

```
<xsl:element name="div">
  <xsl:attribute name="class">chapter</xsl:attribute>
</xsl:element>
```

## XSL constructors: Task 1 discussion

A constructor for an element `<div>` with a `@class` attribute whose value is `"chapter"`:

```
<xsl:element name="div">

    [What can go in this location?]

  <xsl:attribute name="class">chapter</xsl:attribute>

    [What can go in this location?]

</xsl:element>
```

## XSL constructors: Task 1 discussion

A constructor for an element `<div>` with a `@class` attribute whose value is `"chapter"`:

```
<xsl:element name="div">

    [Only attribute constructors can go here.]

<xsl:attribute name="class">chapter</xsl:attribute>

    [Lots of things can go here, including
    <xsl:apply-templates>, other attribute
    constructors, and other element
    constructors.]

</xsl:element>
```

## XSL constructors: Task 2

- **When would I ever actually use this?**
- We've already come across a perfect use-case in the Hamlet transformation:

```
<xsl:template match="head">
  <h2>
    <xsl:apply-templates/>
  </h2>
</xsl:template>
```

- This matches `<head>` elements for both **Act** headings and **Scene** headings.
- What's the problem with it?

- Open hamlet.xml and the example_01.xsl file.
- Set up a transformation scenario (elicit the steps).
- Do the transform, and show the problem.
- Discuss possible solutions. Arrive at the counting of ancestor divs.
- Elicit the initial fix, which would use h{count(ancestor::div)+1}.
- Once that's working, elicit the problem with the general case (h7 etc.).
- Elicit the solution to it (h{min(count(ancestor::div)+1), 6)}.