Schema structure

- Elements are defined by
  - `<element name="…" type="…" minOccurs="…" maxOccurs="…">`
    - *name* refers to the tag.
    - *type* can be custom-defined or one of the standard types. Common predefined types include *string*, *integer* and *anyURI*.
    - *minOccurs* and *maxOccurs* specify how many occurrences of the element may appear in an XML document. *unbounded* is used to specify no upper limits.
- Example
  - `<element name="title" type="string" minOccurs="1" maxOccurs="1"/>`

Sequences

- Sequences of elements are defined using a *complexType* container.

```
<complexType>
   <sequence>
      <element name="title" type="string"/>
      <element name="author" type="string"
               maxOccurs="unbounded"/>
   </sequence>
</complexType>
```

- Note: Defaults for both *minOccurs* and *maxOccurs* are 1

Nested Elements

- Instead of specifying an atomic type for an element as an attribute, its type can be elaborated as a structure. This is used to correspond to nested elements in XML.

```
<element name="uct">
    <complexType>
        <sequence>
            <element name="title" type="string"/>
            <element name="author" type="string"
                     maxOccurs="unbounded"/>
        </sequence>
    </complexType>
</element>
```

Extensions

- Extensions are used to place additional restrictions on the content of an element.
  - Content must be a value from a given set:

```
<element name="version">
    <simpleType>
        <restriction base="string">
            <enumeration value="1.0"/>
            <enumeration value="2.0"/>
        </restriction>
    </simpleType>
</element>
```

  - Content must conform to a regular expression:

```
<element name="version">
    <simpleType>
        <restriction base="string">
            <pattern value="[1-9]\.[0-9]+"/>
        </restriction>
    </simpleType>
</element>
```

Attributes

- Attributes can be defined as part of *complexType* declarations.

```
<element name="author">
   <complexType>
      <simpleContent>
         <extension base="string">
            <attribute name="email" type="string"
                       use="required"/>
            <attribute name="office" type="integer"
                       use="required"/>
            <attribute name="type" type="string"/>
         </extension>
      </simpleContent>
   </complexType>
</element>
```

Named Types

- Types can be named and referred to by name at the top level of the XSD.

```
<element name="author" type="uct:authorType"/>

<complexType name="authorType">
   <simpleContent>
      <extension base="string">
         <attribute name="email" type="string"
                    use="required"/>
         <attribute name="office" type="integer"
                    use="required"/>
         <attribute name="type" type="string"/>
      </extension>
   </simpleContent>
</complexType>
```

Other Content Models

- Instead of *sequence,*
  - *choice* means that only one of the children may appear.
  - *all* means that each child may appear or not, but at most once each.

Many more details about content models can be found in specification!

Schema Namespaces

- Every schema should define a namespace for its elements, and for internal references to types
  -
```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.uct.ac.za"
        xmlns:uct="http://www.uct.ac.za">

<element name="author" type="uct:authorType"/>

<complexType name="authorType">
   <simpleContent>
      <extension base="string">
         <attribute name="email" type="string"
                    use="required"/>
         <attribute name="office" type="number"
                    use="required"/>
         <attribute name="type" type="string"/>
      </extension>
   </simpleContent>
</complexType>

</schema>
```

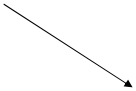
XPath Shorthand

Expression	What it selects in current context
title	"title" children
*	All children
@office	"office" attribute
author[1]	First author node
/uct/title[last()]	Last title within uct node at top level of document
//author	All author nodes that are descendent from top level
.	Context node
..	Parent node
version[number]	Version nodes that have "number" children
version[number='1.0']	Version nodes for which "number" has content of "1.0"

Creating Element nodes

- *element* is replaced by an XML element with the indicated tag.
  - Example:
    - `<element name="dc:publisher">UCT</element>`

```
<xsl:template match="uct:uct">
   <uwc:uwc>
      <xsl:element name="uwc:title">
      </xsl:element>
   </uwc:uwc>
</xsl:template>
```

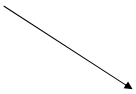


```
<uwc:uwc
     xmlns:uwc='http://www.uwc.ac.za'
     xmlns:uct='http://www.uct.ac.za'
>
   <uwc:title/>
</uwc:uwc>
```

Creating Text nodes

- *text* is replaced by the textual content.
  - Example:
    - `<text>1.0</text>`

```
<xsl:template match="uct:uct">
   <uwc:uwc>
      <xsl:element name="uwc:title">
         <xsl:text>test XML document</xsl:text>
      </xsl:element>
   </uwc:uwc>
</xsl:template>
```

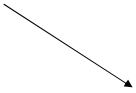


```
<uwc:uwc
     xmlns:uwc='http://www.uwc.ac.za'
     xmlns:uct='http://www.uct.ac.za'
>
   <uwc:title>test XML document</uwc:title>
</uwc:uwc>
```

Element and Text Shorthand

- Elements and text nodes can usually be included directly in templates.
  - Example
    - Instead of `<element name="xxx"/>`
    - Use `<xxx/>`

```
<xsl:template match="uct:uct">
   <uwc:uwc>
      <uwc:title>
         test XML document
      </uwc:title>
   </uwc:uwc>
</xsl:template>
```

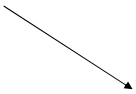


```
<uwc:uwc
     xmlns:uwc='http://www.uwc.ac.za'
     xmlns:uct='http://www.uct.ac.za'
>
   <uwc:title>test XML document</uwc:title>
</uwc:uwc>
```

Copying values across

- *value-of* is replaced with the textual content of the nodes identified by the XPath expression.
  - Example:
    - `<value-of select="uct:title"/>`

```
<xsl:template match="uct:uct">
   <uwc:uwc>
      <uwc:title>
         <xsl:value-of select="uct:title"/>
      </uwc:title>
   </uwc:uwc>
</xsl:template>
```



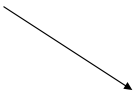
```
<uwc:uwc
      xmlns:uwc='http://www.uwc.ac.za'
      xmlns:uct='http://www.uct.ac.za'
>
   <uwc:title>test XML document</uwc:title>
</uwc:uwc>
```

Applying Templates Explicitly

- *apply-templates* explicitly and recursively applies templates to the specified nodes.
  - Example:
    - `<apply-templates select="uct:version"/>`

```
<xsl:template match="uct:uct">
   <uwc:uwc>
      <uwc:title>
         <xsl:value-of select="uct:title"/>
      </uwc:title>
      <xsl:apply-templates select="uct:author"/>
   </uwc:uwc>
</xsl:template>

<xsl:template match="uct:author">
   <uwc:author>
      <xsl:value-of select="."/>
   </uwc:author>
</xsl:template>
```



```
<uwc:uwc
      xmlns:uwc='http://www.uwc.ac.za'
      xmlns:uct='http://www.uct.ac.za'
>
   <uwc:title>test XML document</uwc:title>
   <uwc:author>Pat Pukram</uwc:author>
</uwc:uwc>
```

Calling Templates

- *call-template* calls a template like a function. This template may have parameters and must have a *name* attribute instead of a *match*.
- Example:

```
<call-template name="doheader">
    <with-param name="lines">5</with-param>
</call-template>

<template name="doheader">
    <param name="lines">2</param>
    …
</template>
```

Variables

- *variable* sets a local variable in a template or globally. In XPath expressions, a $ prefix indicates a variable or parameter instead of a node.
  - Example:

```
<variable name="institution">UCT</variable>
<value-of select="$institution"/>
```

- Expressions also can be inserted into attributes in generated nodes. Surround the expression with { and } to tell XSLT it is not a literal string.
  - Example:

```
<place institution="{$institution}"/>
```

Procedural Constructs

- Generate a tree of nodes if a condition holds.
  - `<if test="position()=last()">…</if>`
- Generate a different tree of nodes, depending on which of a number of a conditions holds.
  -
```
<choose>
    <when test="$val=1">…</when>
    <otherwise>…</otherwise>
</choose>
```
- Iterate over a set of nodes matching an expression and generate a tree for each. This has the same effect as *apply-templates*.
  - `<for-each select="uct:number">…</for-each>`