FOR PUBLIC RELEASE

5

The XSLT transformation environment

5 The XSLT transformation environment

Some aspects of our stylesheets are of a global nature that we must consider before we delve into the details of template instruction execution and behaviors. This chapter overviews the aspects of our environment in which we use our XSLT processor from the perspectives of the stylesheet contents, the serialized output (if any), and the operator invoking the processor.

All explicitly declared stylesheets follow a required shape of container and top-level (container children) elements. Methods are also specified for including arbitrary information in a stylesheet file, useful for supplemental information for processing, or as documentation of the stylesheet content.

The stylesheet can declare its desire for values of certain parameters of the output serialization that influence the contents of the reified result node tree.

In addition, there are a number of ways available to communicate with an XSLT processor that is interpreting a stylesheet resource against a source resource. Communication to the processor can be engaged at invocation as well as from the processor during execution.

Finally, this chapter reviews aspects of the transformation environment that cannot be controlled by the operator or the stylesheet. It is important to understand the limitations of what can be asked for or even supported by the XSLT processor.

This chapter includes discussion of the following XSLT instructions regarding the transformation environment in which a stylesheet is used.

Instructions for wrapping the content of a stylesheet are as follows.

- `<xsl:stylesheet>`
  - encapsulates a stylesheet specification.
- `<xsl:transform>`
  - encapsulates a stylesheet specification.

Instructions for serializing the result tree are as follows.

- `<xsl:namespace-alias>`
  - specifies a result tree namespace translation.
- `<xsl:output>`
  - specifies the desired serialization of the result tree.

Instructions for communicating with the operator are as follows.

- `<xsl:message>`
  - reports a stylesheet condition to the operator.
- `<xsl:param>`
  - supplies a parameterized value from the operator.

5.1 Stylesheet basics

5.1.1 The stylesheet document/container element

Two identical and interchangeable choices for the document element are:

- `<xsl:stylesheet>`
- `<xsl:transform>`

They can also be used as a container element for a stylesheet embedded in another context.

- They may use `id="`*`unique identifier`*`"`.
  - It identifies the stylesheet when there are multiple ones from which to choose.
  - The use of this XML `ID` attribute is outside the scope of the Recommendation.
  - It could be used as a fragment identifier by the stylesheet association processing instruction or by other techniques to identify a given stylesheet among many.

Controls available on container element or any literal result element are:

- `exclude-result-prefixes="`*`whitespace-separated-prefixes`*`"`
  - scope of influence is all descendent elements in stylesheet;
  - this declaration indicates which stylesheet namespace prefixes are not expected in the result, thus are not to be included in the stylesheet tree;
    - a list of whitespace-separated namespace prefixes specifies prefixes that are to be explicitly excluded from the stylesheet tree (using `#default` as the name to reference the default namespace, which is sometimes unofficially called the null namespace);
  - user-specified prefixes and associated namespace declarations are often used in XSLT stylesheets (but not desired in the result) for various purposes such as:
    - top-level documentation,
    - embedded structured data,
    - named XSLT constructs;
  - recall that copying an element node from the stylesheet to the result will copy all attached namespace nodes, thus stylesheet namespace declarations can easily end up in the result tree;
    - a stylesheet wrapper-element namespace declaration is typically used for top-level namespace usage, thus the document element of the result will typically end up with the same declarations;
  - this exclusion declaration tells the XSLT processor to not include the specified namespace nodes on descendent nodes of the stylesheet tree;
  - this exclusion declaration has no effect on namespace nodes of the source tree,
- `extension-element-prefixes="`*`whitespace-separated-prefixes`*`"`
  - scope of influence is all descendent elements in stylesheet;
  - this declaration indicates which stylesheet namespace prefixes are instruction prefixes;
    - a list of whitespace-separated namespace prefixes specifying prefixes that are extension namespaces to be recognized by the XSLT processor (using `#default` as the name to reference the default namespace);
  - recall that everything that is not an instruction is considered to be a literal result element;

- elements prefixed with the namespace prefix associated with the XSLT URI are interpreted as instructions;
- this declaration tells the XSLT processor what other prefixes are to be interpreted as instructions because they are extension elements required by the stylesheet;
- the processor need not implement the extension elements (detailed in Chapter 6).

Child elements of the document or container element are referred to as "top-level" elements.

- If present, the following must occur before all other top-level elements:
  - `xsl:import`
    - see Chapter 6.
- If present, the following (listed alphabetically) may occur in any order as top-level elements:
  - `xsl:attribute-set`
    - see Chapter 7,
  - `xsl:include`
    - see Chapter 6,
  - `xsl:key`
    - see Chapter 8,
  - `xsl:decimal-format`
    - see Chapter 8,
  - `xsl:namespace-alias`
    - see this chapter,
  - `xsl:output`
    - see this chapter,
  - `xsl:preserve-space`
    - see Chapter 3,
  - `xsl:strip-space`
    - see Chapter 3,
  - `xsl:template`
    - see Chapter 4.
- The following are used not only as top-level elements, while all others listed above are only used as top-level elements:
  - `xsl:param`
    - see Chapter 6,
  - `xsl:variable`
    - see Chapter 6.

5.1.2 Documenting stylesheets

Because an XSL stylesheet is an XML document —

- XML comments can be used to provide documentation about the stylesheet,
- all XML comments and processing instructions found in an XSL stylesheet are ignored.
  - Note that some XML editing tools may leave processing instructions in files for remembering locations such as the last cursor position.

Adding richly marked up documentation to a stylesheet:

- allows the stylesheet to be run through a documenting stylesheet to extract the documentation in any fashion desired,
- is accomplished by including non-XSLT constructs as top-level elements (children of the stylesheet document element) provided that the default namespace is not used as the namespace for such constructs, as in the following example:

Example 5–1 Using non-XSLT constructs as top-level elements

```
<?xml version="1.0"?>                                    <!--hellodoc.xsl-->
<!--XSLT 1.0 - http://www.CraneSoftwrights.com/training -->

<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
               version="1.0" exclude-result-prefixes="mydoc"
               xmlns:mydoc="http://www.mycompany.com/mydoc">

<xsl:output method="html"/>

<mydoc:para>
The following construct is the root template.
</mydoc:para>

<xsl:template match="/">                                    <!--root rule-->
     <b><i><u><xsl:value-of select="greeting"/></u></i></b>
     <?test a processing instruction here?>
</xsl:template>

</xsl:transform>
```

Note the use of `exclude-result-prefixes=` in the document element above to tell the XSLT processor to not emit a namespace declaration for the prefix of the documentation namespace —

- if the stylesheet writer knows that namespace will never be needed in the result;
- because the XSLT processor doesn't know when creating the document element node of the result tree whether the namespace will ever be needed in the instance, so by default the declaration is emitted.

5.1.3 Namespace protection

Some special concern regarding the use of namespaces are as follows.

- The "transformation by example" paradigm utilizes literal result elements.
  - It represents result tree element nodes with associated attribute nodes.
    - An element is written with associated attributes in a template in the stylesheet and can use —
      - the default namespace,
      - a namespace prefix and associated namespace URI.
    - An alternative described later is the use of XSLT instructions to synthesize result tree nodes.
- Some namespaces can be sensitive in the document processing environment;
  - this includes automatically-triggered platform services;
  - for example: digital signature processing.
- If the result tree requires a sensitive namespace, the stylesheet can't use the namespace in a literal result element —
  - to produce an XSLT script as the output of translation;
    - the XSLT processor would incorrectly interpret the result vocabulary as input,
  - to use a platform service for the output of translation;
    - the stylesheet use of the URI would incorrectly trigger the service.
- The `<xsl:namespace-alias` ***`attributes`***`/>` top-level element can be used, which:
  - is an instruction to translate a namespace prefix in the stylesheet into another namespace prefix when used in the result;
  - with attribute `stylesheet-prefix="`***`prefix`***`"`:
    - specifies the prefix used in the stylesheet tree that is being added to the result tree by the stylesheet,
      - has no influence or recognition in the source tree,
  - with attribute `result-prefix="`***`prefix`***`"`:

- specifies the prefix of the stylesheet tree whose URI is to be used for the result tree prefix,
- must have this attribute declared in the stylesheet even if no element in the stylesheet uses the prefix.

Note in the example below how the XSLT namespace URI cannot be used for the declaration for the `xslo` prefix, otherwise the `xslo` prefixed elements would be interpreted as XSLT instructions.

Example 5–2 A stylesheet that produces a stylesheet

```
<?xml version="1.0"?>                                       <!--xsl.xsl-->
<!--XSLT 1.0 - http://www.CraneSoftwrights.com/training -->

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xslo="any-URI" version="1.0">

<xsl:output indent="yes"/>

<xsl:namespace-alias stylesheet-prefix="xslo"
                     result-prefix="xsl"/>

<xsl:template match="/">                                  <!--root rule-->
  <xslo:stylesheet version="1.0">
    <xslo:template match="/">
      <html>
        <p>Hello world</p>
      </html>
    </xslo:template>
  </xslo:stylesheet>
</xsl:template>

</xsl:stylesheet>
```

- When this particular stylesheet is run with itself (or any XML file) as the source, the XSLT processor will assign the “`xslo:`” prefix’s URI used in the result tree with the URI for the “`xsl`” prefix as indicated in the `<xsl:namespace-alias>` instruction, thus using the XSLT URI when the result tree is serialized as XML markup:

Example 5–3 A stylesheet produced by a stylesheet

```
<xslo:stylesheet version="1.0"
xmlns:xslo="http://www.w3.org/1999/XSL/Transform">
<xslo:template match="/">
<html>
<p>Hello world</p>
</html>
```

```
</xslo:template>
</xslo:stylesheet>
```

- When run with itself (or any XML file) as the source, the output will be:

Example 5–4 The result produced by the produced stylesheet

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<p>Hello world</p>
</html>
```

5.2 Communicating with the XSLT processor

5.2.1 Serializing the result tree

The `<xsl:output>` top-level element:

- is a request to serialize the result tree as a sequence of bytes.
  - The XSLT processor *may* choose to respect the request, but is not obliged.

All attributes of `<xsl:output>` are optional.

- The `method="`*`method-indication`*`"` attribute may take these values:
  - `method="html"`
    - uses the HTML vocabulary and SGML markup conventions, namely:
      - empty elements,
      - attribute minimization,
      - built-in character entity referencing (ISO Latin 1),
      - the entire set of HTML conventions (you cannot selectively turn on only a subset of them);
        - all conventions are used according to common practice,
    - is considered the default in certain result tree conditions;
      - the name of the document element node is HTML (case insensitive);
      - the null namespace URI is used for the name (i.e.: there is no namespace prefix);
      - any preceding text nodes contain only whitespace,
  - `method="xml"`
    - uses arbitrary vocabulary and XML markup conventions, namely:
      - empty elements,
      - built-in character entity referencing,
    - is the default when the default isn't HTML,
  - `method="text"`
    - uses no vocabulary and no lexical or syntactic conventions,

        - serializes only the text nodes of every element in the result tree,
        - outputs all characters in clear text (no entities of any kind),
    - is never the default,
  - `method="`*`prefix:processor-recognized-method-name`*`"`
    - uses the prefix defined by `xmlns:`*`prefix`*`="`*`processor-recognized-URI-reference`*`"`;
    - uses lexical and syntactic conventions recognized by the XSLT processor;
      - in particular, serialization can be arbitrary (it is out of the scope of XSLT);
    - is never the default.
- Attributes related to the method are as follows:
  - `version="`*`numeric-version`*`"`
    - specifies the version of the output method,
  - `omit-xml-declaration="yes"` or `omit-xml-declaration="no"`
    - specifies the absence or presence of the XML declaration (if the result tree represents a document entity) or the text declaration (if the result tree represents an external general parsed entity),
  - `standalone="yes"` or `standalone="no"`
    - specifies the presence or absence of a standalone document declaration,
  - `doctype-system="`*`system-identifier`*`"`
    - specifies the system identifier to use in the `DOCTYPE` declaration,
  - `doctype-public="`*`public-identifier`*`"`
    - specifies the public identifier to use in the `DOCTYPE` declaration,
    - requires `doctype-system=` to also be specified if the output method is XML.
- Attributes related to the serialized markup syntax are as follows:
  - `indent="yes"`
    - asks the XSLT processor (at its discretion) to indent the result "nicely" with additional whitespace when using the `xml` method;
      - this may have implications for the downstream parsing processes if the whitespace is considered significant,
  - `cdata-section-elements="`*`list-of-element-type-names`*`"`
    - gives a whitespace separated list of element types possibly used in the result,
    - specifies those result tree elements whose text content is serialized within a `CDATA` section.
- Attributes related to the encoding are as follows:
  - `encoding="`*`encoding`*`"`
    - requests (if supported by the processor) the character set encoding output of the emitted result tree,
    - has the value which should match the `encoding=` pseudo-attribute described by the XML Recommendation for the XML declaration,
  - `media-type="`*`media-type`*`"`
    - specifies the MIME content type (without specifying the `charset` parameter).

5.2.2 Illustration of output methods

Consider a simple XML file `nodein.xml` created using the 8-bit ISO character set for Western European languages Latin–1, a.k.a. ISO–8859–1 (note the copyright symbol seen here is encoded in the file using the hexadecimal character `0xA9`):

Example 5–5 An XML source file with characters sensitive to processing

```
<?xml version="1.0" encoding="iso-8859-1"?>
<p>Test with © and &lt; and &amp; in it</p>
```

Figure 5–1 illustrates the node tree that is created by the XSL processor.

Note how the markup used to represent the sensitive XML characters is lost. The node tree shown would also be created identically by the following markup:

Example 5–6 The same information using a `CDATA` section

```
<?xml version="1.0" encoding="iso-8859-1"?>
<p><![CDATA[Test with © and < and & in it]]></p>
```

All character values in text nodes are maintained as UCS–2 (Universal Character Set — Two Octet) characters. The UCS character set is a 32-bit (4 octet) repertoire with a 16-bit (2 octet) repertoire subset (equivalent to Unicode) that can be serialized as either 16-bit (2 octet) characters or, using an encoding called UTF–8, as a sequence of 8-bit (1 octet) characters.

Utilizing an extension element defined in XT providing for multiple result trees, one can copy the source node tree to each of three result

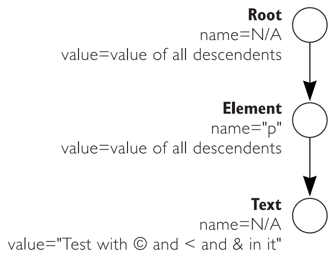
Root
name=N/A
value=value of all descendents
Element
name="p"
value=value of all descendents
Text
name=N/A
value="Test with © and < and & in it"

Figure 5–1 Illustration of Node Tree Characters

trees, such that each result tree is identical to the source tree, and interpret each result tree differently:

Example 5–7 Emission of the source tree using three different output methods

```
<?xml version="1.0"?>                                       <!--nodeout.xsl-->
<!--XSLT 1.0 - http://www.CraneSoftwrights.com/training -->
<!--XT (see http://www.jclark.com/xml/xt.html)-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0"
                xmlns:xt="http://www.jclark.com/xt"
                extension-element-prefixes="xt">

<xsl:template match="/">
  <xt:document method="xml"  href="nodeout.xml"
               omit-xml-declaration="yes">
    <xsl:copy-of select="."/>
  </xt:document>
  <xt:document method="html" href="nodeout.htm">
    <xsl:copy-of select="."/>
  </xt:document>
  <xt:document method="text" href="nodeout.txt">
    <xsl:copy-of select="."/>
  </xt:document>
</xsl:template>

</xsl:stylesheet>
```

There is a nuance here regarding the use of the extension element: the `<xt:document>` element creates a separate result tree, and is not a result tree element itself that resides in a single "master" result tree as might be evident.

The use of `method="xml"` emits the same nodes using the UCS characters of the text nodes while using the built-in XML entities where necessary:

Example 5–8 XML output method emission of sample instance

```
<p>Test with © and &lt; and &amp; in it</p>
```

- Note the two-character UTF–8 hexadecimal representation of the copyright symbol is `0xC2 0xA9` which would both be revealed in a non-UTF–8 presentation environment such as an ISO–8859–1 Latin–1 environment as follows:

  ```
  <p>Test with Â© and &lt; and &amp; in it</p>
  ```

The use of `method="html"` recognizes known built-in HTML entities and uses the entity references where necessary:

Example 5–9 HTML output method emission of sample instance

```
<p>Test with &copy; and &lt; and &amp; in it</p>
```

The use of `method="text"` ignores all element start and end tags and puts out the UCS characters of all the text nodes while not using any built-in entities:

Example 5–10 Text output method emission of sample instance

```
Test with © and < and & in it
```

- Note again in a non-UTF–8 environment this text file would appear as two characters as in the ISO–8859–1 Latin–1 environment:

  ```
  Test with Â© and < and & in it
  ```

5.2.3 Communicating with the outside environment

These instructions are used for communication between the stylesheet and the XSLT processor and the operator:

- stylesheet to operator: `<xsl:message>`
  - it contains an arbitrary message such as —
    - status of progress,
    - content violation;
  - the specific mechanism of communication is not standardized;
  - the processor may choose to not support relating the message,
  - the content is any template (static or calculated),
  - this instruction can contain the `terminate="yes"` attribute —
    - which gives an instruction to stop any further processing of the stylesheet and source files,
  - this instruction allows the stylesheet to report on semantic validation;
    - when content has been detected as being incorrect, messages can report problems to the operator;
    - structural well-formedness correctness has already been determined by the XML processor inside the XSLT processor;
    - stylesheet could also use XPath to determine structural validity if the XSLT processor does not use a validating XML processor;
  - this instruction allows the stylesheet to report progress when manipulating large data sets,
- operator to stylesheet: `<xsl:param>`

  - it provides an invocation-time parameterized value for a globally scoped bound variable;
  - the specific mechanism of communication is not standardized;
  - the processor may choose to not support value specification;
  - a default value can be specified should no value be supplied at invocation,
- processor to stylesheet:
  - to obtain the value of a system property, use:

    `system-property('`*`prefix`*`:`*`property-name`*`')`

  - use XSLT namespace to indicate reserved system properties:
    - `xsl:version`
      - returns a decimal number (not a string) of the XSLT processor's implementation level in order to test the level of functionality for a given stylesheet;
    - `xsl:vendor` and `xsl:vendor-url`
      - each returns a string indicating, respectively, the name and URL (Uniform Resource Locator — RFC–1738/RFC–1808/RFC–2396) of the vendor of the executing XSL processor;
  - use other namespaces to indicate extension system properties:

    `xmlns:`*`prefix`*`="`*`processor-recognized-URI-reference`*`"`

    `system-property('`*`prefix`*`:`*`property-name`*`')`

    - the processor returns the empty string for an unrecognized property.

The following example illustrates how to tell the operator the stylesheet uses features not supported by the processor.

Example 5–11 An example of utilizing available system properties

```
  <xsl:choose>
    <xsl:when test="system-property('xsl:version') >= 2.0">
      <xsl:feature-of-2.0/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:message terminate="yes">
Sorry, this stylesheet requires XSLT 2.0
Complain to: '<xsl:value-of
                      select="system-property('xsl:vendor')"/>'
at '<xsl:value-of select="system-property('xsl:vendor-url')"/>'.
      </xsl:message>
    </xsl:otherwise>
  </xsl:choose>
```

5.2.4 Uncontrolled processes

There is no recommendation-based user or stylesheet control over or communication available regarding the following processes implemented by the XSLT processor.

- Result tree attribute order:
  - the XSLT processor may choose to serialize attribute nodes found in the result tree in any order.
- Result tree serialization instance markup:
  - the XSLT processor may choose any way it desires to serialize the content of text nodes when the stylesheet does not instruct a given element to be emitted as a CDATA section —
    - using XML built-in character entities for markup-sensitive characters,
    - using numeric character entities for markup-sensitive characters or characters not present in the encoding character set,
    - using piecemeal CDATA sections;
  - any original markup syntax from the source file is lost when the source file is abstracted into the source node tree;
  - other than an entire element emitted as a CDATA section, there is no control available in the stylesheet over which serialization methods are used for text content.
- Result tree construction:
  - the stylesheet writer is responsible for dictating the final content of the result tree;
  - the XSLT processor can use any means to effect the final result as described in the Recommendation without necessarily implementing the prose description found therein;
  - the side-effect free nature of the XSLT design (including the inability to change the value of bound variables) allows an XSLT processor to process portions of the input in parallel and combine the intermediate results into the single final result tree;
  - the of XSLT allows the processor to choose to not preserve the result node tree when serializing the transformed information to an output instance, thus the result may never actually exist as a complete tree within the processor.