Anatomy of an XSLT Stylesheet

- Overview of XML
- Introduction to XSLT and XPath
- Nodes
- Document Order
- Converting XML to HTML
XSLT, the extensible stylesheet language for transformations, is a language that provides the mechanism to transform and manipulate XML data. It is impossible to discuss XSLT without a reference to XML (extensible markup language). XML is a related W3C (World Wide Web Consortium) standard, and is the basis for standard information interchange. XML provides structure to information, and XSLT, along with another related standard, XPath (XML path language), provides the means to extract, restructure, and manipulate that information.

XSLT has the same cross-platform functionality found in XML because it is written according to the same rules. If you are an experienced markup technologist proficient in XML, you will be able to quickly maximize the uniquely powerful language of XSLT. In fact, by the end of this chapter, you will be able to write XSLT stylesheets that perform basic transformations from XML to HTML.

This chapter provides explanations and analogies for using XML, XSLT, and XPath. Any prior knowledge you have of XML markup will greatly speed up the learning process; however, a brief description is presented in the following section as a review.
1.1 What Is Markup?

Without retelling the story of how markup evolved or what it is, it is important to begin with a conceptual understanding of XML. This book is not intended to provide an explanation of the complete syntax and usage of XML, however, there are a few concepts that are worth reviewing. XML is a markup language that is a derivation—or flavor—of SGML (Standard Generalized Markup Language). You may be more familiar with HTML, a popular example of SGML that is used to mark up content for presentation on the Web.

Markup is made up of *tags*, which describe and separate the contents of an XML document instance from the presentation, style, and format of the document. The tags can be thought of as hooks, or handles, by which all material they contain—the text or data—can be accessed or identified.

The objects that the tags contain are called *elements*. Elements are the main components of an XML document, and can be identified by their element-type name, which is contained in both the start tag and the end tag, as shown below:

```xml
<para>This is a paragraph</para>
```

The paragraph tags, shown here as `<para>` and `</para>`, are used to separate the contents of the the paragraph from the other paragraphs and elements in the XML document. Notice that we have made up our own element-type names here (instead of using the old HTML `<p>` tags). XML allows you to create your own tag names, adding infinitely more functionality and precision of markup than HTML provides. As another feature of XML, elements can contain text, as shown above, and can also contain other elements, for example:

```xml
<para>This is a <index>paragraph</index></para>
```

Elements inside elements introduces the concept of nesting, and of relationships between the elements that can be addressed directly. Addressing can now be done based on the location and relationship of an element inside another, as in "all the index elements *inside* a paragraph." Nesting elements also creates a structure that can be represented as a tree.

1.1.1 Markup Grows on Trees

When explaining markup, the image of a tree is often used to illustrate concepts, designs, processes, and other ideas, both tangible and intangible. The
1.1 What Is Markup?

root, most especially, has a long signification throughout the history of human thought and philosophy as the base, or beginning. The root in XML is similar. From the root of an XML document, a tree-like structure emerges, which can be navigated and referenced with great precision. This precision is essential for working with XML. The access to and use of the entire structure of an XML document are inseparable from its representation as a tree-like structure.

Understanding the tree structure of an XML document is crucial to navigating it. The tree in markup, especially in XML, is highly abstracted and stylized to convey specific characteristics of the markup language. It hangs from the bottom, as shown in the example of a book in Figure 1–1.

![Figure 1–1 Tree representation of a basic XML document.](image)

XML has many other aspects, enough for countless other books! Because we are dealing with XSLT as it relates to XML, we will not cover all the concepts that XML brings, only those directly related to using XSLT.
Chapter 1 Anatomy of an XSLT Stylesheet

1.2 What Is XSLT?

In its most basic sense, XSLT is XML. The familiar structure of markup, using less-than and greater-than symbols (“<” and “>”), as seen, for instance, in `<xsl:stylesheet>`, makes its syntax readily identifiable.

There are several benefits to thinking of XSLT as an XML document instance. Of course, aside from the familiar tagging structure, it is important to have specifications that conform to the same syntax, are platform-independent, and can be parsed by the same basic technology.

Another benefit is the notion of well-formedness, which allows the structuring of XSLT stylesheets to proceed without a particular DTD. The importance of well-formedness for an XSLT stylesheet cannot be emphasized enough—both for the XSLT stylesheet to successfully parse when initially read by an XSLT processor and to be readily understood, debugged, or adjusted.

XSLT is used to transform XML documents into other XML documents. XSLT processors parse the input XML document, as well as the XSLT stylesheet, and then process the instructions found in the XSLT stylesheet, using the elements from the input XML document. During the processing of the XSLT instructions, a structured XML output is created. XSLT instructions are in the form of XML elements, and use XML attributes to access and process the content of the elements in the XML input document.

XSLT is not generally used for formatting. There is a separate specification for formatting from the W3C called XSL, which is generally called XSL FO (formatting objects). XSLT can affect formatting if, for instance, the XSLT stylesheet is designed to output HTML tags for display in a browser, but this is only a small fragment of its capabilities.

---

1. XSLT does not conform to a specific Document Type Definition (DTD), however the basic set of elements to be supported by an XSLT processor is described in a non-normative DTD in the XSLT specification.

2. The XSL specification can be found at http://www.w3.org/TR/xsl/.
1.3 What Is XPath?

XSLT is rarely discussed without a reference to XPath. XPath is a separate recommendation from the W3C that uses a simple path language to address parts of an XML document. Although XPath is used by other W3C recommendations, there is hardly a use for XSLT that does not involve XPath. Generally speaking, XSLT provides a series of operations and manipulators, while XPath provides precision of selection and addressing.

1.3.1 The XSLT Stylesheet

This structured hierarchy of elements for the book in Figure 1–1 is a useful way to begin to understand how XSLT stylesheets work. Figure 1–2 represents the same tree structure, only here it reflects some basic components of an XSLT stylesheet.

---

3. Both XSLT and XPath became full recommendations on the same day, November 16, 1999.
Chapter 1 Anatomy of an XSLT Stylesheet

The use and explanation of these components will be provided in more detail later in this chapter, but what is important to stress here is that XSLT is XML, and has the same overall structure.

1.4 XSLT Stylesheet Concepts

XSLT stylesheets are best understood according to their structure and the named elements within them. It has always been a hallmark of markup languages that there be a diligent attempt at human-readability for the element-type names and, where possible, other components. With XSLT, this has been fairly well achieved, making it easier to learn and understand XSLT stylesheets.

Let’s compare the XML tree structure of a book with that of an XSLT stylesheet, shown side-by-side in Figure 1–3.

![XML and XSLT trees comparison]

If we rendered the XSLT side of this diagram as a stylesheet, it would look like Example 1–1.

This example shows the basic components of most XSLT stylesheets. The `<xsl:stylesheet>` element contains two other elements, an `<xsl:output>` element and an `<xsl:template>` element (sometimes called a template rule).

The HTML `<p>` tag simply sends the same tag to the output, as does the `<hr>` tag.
Another component in our example is the `<xsl:apply-templates>` element, which is inside the `<p>` tags.

Using the `<book>` XML input sample would generate the output shown in Example 1–2.

**Example 1–1 :** XSLT stylesheet as an XML document.

```xml
<?xml version="1.0"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
    <xsl:output method="html" />

    <xsl:template match="topic">
        <p>
            <xsl:apply-templates />
        </p>
        <hr/>
    </xsl:template>
</xsl:stylesheet>
```

**Example 1–2 :** Processing a topic.

**INPUT:**

```xml
<book>
    <intro/>
    <chapter>
        <section>
            <topic source="song">Xanadu</topic>
            <topic>Topic 2</topic>
        </section>
    </chapter>
</book>
```

**OUTPUT:**

```xml
<p>Xanadu</p>
<hr/>
<p>Topic 2</p>
<hr/>
```
Notice the structure of the XSLT stylesheet in Example 1–1 that contributed to this output. The \(<xsl:template>\) element found a match on a \(<topic>\) in the input XML document. This was replaced by the contents of that \(<xsl:template>\) element—in this case, the \(<p></p>\) and \(<hr/>\) elements. Then the content of the \(<topic>\) was sent to the output, enclosed within the \(<p>\) and \(</p>\) open and close tags. This was done by the \(<xsl:apply-templates>\) element, which is contained within the \(<p>\) tag in the stylesheet.

Notice also that the generated output is not well-formed XML. XSLT processors generate XML documents, but do not parse the output document.

1.4.1 Using XSLT to Convert XML to HTML

Let’s use some simple XSLT to transform the elements in an XML document to HTML. As an example, we will use the common image of a year, subdivided by a loose notion of an agricultural calendar with planting, harvest, seasons, and months. Example 1–3 illustrates how markup might be used to describe a year.

Suppose we need to render our year for display in a conventional Web browser. Conversion to HTML is a frequent task for XSLT stylesheets. We’ll make this a simple transformation, using the HTML unordered list format \(<ul></ul>\) for displaying the \(<month>\)s.

Using the stylesheet in Example 1–4, we create a simple list of the months with the HTML list item tags \(<li></li>\) for the output document.

Again, consider the XML document instance structure of this XSLT stylesheet. Taking for granted the required stylesheet components that will be discussed later, the two template rules that remain are simple. The first matches the \(<year>\) (using the \(match=\"year\"\) attribute) and replaces it with an unordered list tag \(<ul></ul>\). The \(<ul>\) element is a child of the \(<xsl:template>\) element. Then, within the \(<ul>\) is the

---

4. Of course, variations in planting and harvest, as well as in seasons, are widespread (with the changing global climate, your experience of the weather might be alarmingly different!), so this is a generalized presentation, based primarily on the seasons and weather of the Northern Hemisphere (though our Iowa friends might take us to task on some points!).
Example 1–3: Marking up a year with XML.

```xml
<?xml version="1.0"?>
<year>
  <planting>
    <season period="spring">
      <month>March</month>
      <month>April</month>
      <month>May</month>
    </season>
    <season period="summer">
      <month>June</month>
      <month>July</month>
      <month>August</month>
    </season>
  </planting>
  <harvest>
    <season period="fall">
      <month>September</month>
      <month>October</month>
      <month>November</month>
    </season>
    <season period="winter">
      <month>December</month>
      <month>January</month>
      <month>February</month>
    </season>
  </harvest>
</year>
```

The `xsl:apply-templates` instruction—`<xsl:apply-templates>`—to process any children of `<year>`. The `<xsl:apply-templates>` instruction element basically tells the processor to look for an `<xsl:template>` for each child of the `<year>`, recursively addressing each child of a child until all the descendants are processed. If the processor finds a rule for an element, it will follow the instructions in that template rule to process the node. If it does not, it will continue working down through the descendants until it reaches a text node. At this point, the text is sent to the output.

The second template rule matches each month in the input document. Each `<month>` tag is replaced by the contents of the template rule (which can be called simply the template), in this case, a list item in HTML.
Chapter 1  Anatomy of an XSLT Stylesheet

Example 1–4: Basic stylesheet for unordered lists.

```
<?xml version="1.0"?>
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
   <xsl:output method="html" />
   <xsl:template match="year">
     <ul>
       <xsl:apply-templates />
     </ul>
   </xsl:template>
   <xsl:template match="month">
     <li>
       <xsl:apply-templates />
     </li>
   </xsl:template>
</xsl:stylesheet>
```

(<li></li>). Then, the text node child of each <month> is sent to the output with <xsl:apply-templates>. To be proper HTML, the output would still need a few things, but it will actually display in most browsers as shown in Figure 1–4.

Example 1–5 shows the resulting HTML when an examination of the source for this file is made. Notice that the file does not contain the normal <html> or <body> tags that are used in most HTML files.

Without having element rules to match on the <harvest>, <planting>, and <season> elements, their tags will not be output to the result, nor will the <html> and <body> tags that are required by HTML (but most browsers would properly display this as an unordered list, anyway). It is possible to do quite a few easy XML-to-HTML transformations in this way.

During our examples we have skipped over some very basic concepts that are crucial to understanding XSLT stylesheets. With this basic example of converting XML to HTML, we will now discuss the concepts and terminology that apply to all XSLT stylesheets.
Figure 1–4  Web view of output from basic stylesheet for unordered lists.

Example 1–5:  HTML output from basic stylesheet for unordered lists.

```html
<ul>
  <li>March</li>
  <li>April</li>
  <li>May</li>
  <li>June</li>
  <li>July</li>
  <li>August</li>
  <li>September</li>
  <li>October</li>
  <li>November</li>
  <li>December</li>
  <li>January</li>
  <li>February</li>
</ul>
```
1.5 Terminology for XSLT

Now that we have seen some very simple XSLT stylesheets, it is important to understand the way in which both XSLT and XPath refer to their parts. For instance, while you may be familiar with “root elements,” the distinction between them and the “root” or “document root” might not be clear. Similarly, the idea of nodes and document order may be generally clear, but their precise applications deserve attention.

1.5.1 The Root of the Matter

The definition of root, shown below, supplies key concepts for working with XSLT stylesheets and other XML document instances.

\[
\text{root}: \text{A unique node or vertex of a graph from which every other node can be reached.}
\]

[First computer usage] Formally, a tree is a set of nodes connected by branches such that there is one and only one way of going from one node to another via branch connections, and which has a distinguished node called the root node. W. C. Gear, *Intro. Computer Sci.*, v. 282, 1973 (OED, 2000, XIV: 88)

This definition introduces three key components for working with XSLT: nodes, directional navigation (though, as we’ll see, when using XPath we are not limited to “one and only one way of going from one node to another”), and the uniqueness of the root node. We will address each of these in turn in the following sections, beginning—naturally—with the root.

In both XML and XSLT, the distinction between the “root” and the "root element" is often confused. The simplest way to untangle these is to look at both terms separately.

*Root—\text{that from which all else comes, on which all else is predicated, and from which every other node can be reached.}*

The root is not an element, it is a container for the XML document. Sometimes the root is also called the document root or the root node. The reference is to the document itself—the object that contains all the elements,
attributes, comments, text, and so on. The root is not a child of any other node.

Root Element—the first element in a document, also known as the document element.

The root element is the single element of which all other elements in the XML document instance are children or descendants. The root element is itself a child of the root. The root element is also known as the document element, because it is the first element in a document, and it contains all other elements in the document.5

In XSLT, the root is the XSLT stylesheet. It contains the XML declaration, `<xml version="1.0"?>`, as its first child. Following that is the document element of the XSLT stylesheet, which can be either `<xsl:stylesheet>` or `<xsl:transform>`. The XML declaration and the document element are the only direct children of the XSLT stylesheet. All other parts of an XSLT stylesheet are contained within the XSLT document element.

Because of the general confusion between the root and the root element, we will generally refer to the root element as the document element.

When an XSLT stylesheet refers to the root of the XML document instance it is processing, the symbol “/” is used. This symbol, called a token, is similar in meaning to the UNIX use, which refers to the “root” on a server. In fact, the entire syntax for referencing parts of the tree descended from the root in an XML document instance is very much like the syntax used in UNIX or MS-DOS to refer to directories and subdirectories. The / symbol and other tokens are discussed in the XPath introduction in Chapter 4.

1.5.2 Branching Out: Nodes

In XML, any point you can identify in the document’s tree structure is a node. XSLT and XPath, when used effectively, permit direct access to any node in the tree. If you refer to “this paragraph,” it’s a node. If you refer to “that element,” “that attribute,” and so on, they are all nodes.

---

5. It is the document element that is referenced as the document type in a Doctype Declaration. For example, in `<!DOCTYPE myelement SYSTEM "mydtd.dtd">`, myelement is the document element.
Chapter 1  Anatomy of an XSLT Stylesheet

The terminology of nodes is used throughout both XSLT and XPath and has crucial import in understanding and accessing each object in a document.

node: The point of a stem from which the leaves spring…a point or vertex of a network or graph. (OED, 2000, X: 459-460)

Let’s reconsider the diagram of the book tree, shown again in Figure 1–5, this time with some of the nodes identified. The book, chapters, sections, and so on are all element nodes. The text contained in each topic is a text node.

Suppose we used attributes to distinguish between types of topics, shown in this example as the attribute “Source” with a value of “song.” This is important with the word “Xanadu,” where it can mean a mythical place enshrined in the flowing words of a Coleridge poem, a pop-music song by Olivia Newton-John, or even a whimsical name used by locals for Sun

Figure 1–5  Nodes in an XML tree structure.
Microsystems’ new campus just northwest of Boston. We could use attributes in a `<topic>` element to mark “Xanadu” to clarify each kind of use. In this case, each attribute along with its value would comprise a node.

There are seven kinds of nodes in XSLT, but we will focus first on the element node as most common to both XML and XSLT stylesheets. The other six node types (that the designers of XSLT thought were significant) are discussed throughout the remainder of this book.

The nodes extending from, or “under,” a chapter would be a set of nodes that is best understood as a node branch. A node branch is any logical structure consisting of a node and its descendants, sometimes referred to as a subtree. A little “pruning” of our tree terminology is implied by the notion of nodes and node branches. You will find that XML terminology does not include “leaves” or “branches.” Nodes and node branches are the closest correlations to these concepts.

Consider again the diagram of a book, shown in Figure 1–6, this time with some of its parts also identified in terms of a node branch.

Figure 1–6 Nodes and node branches in an XML tree.
You would use XSLT and XPath to access any one of these nodes or a combination of them, as well as other types of nodes not shown here. The nodes descended from and including the “section” on the far left are a node branch.

Node-sets, on the other hand, are described in the XPath specification as “an unordered collection of nodes without duplicates.” They are the set of nodes returned by an expression (discussed in Chapter 3), regardless of the location of the nodes in the tree or branch of the XML instance. For example, the set of two section elements can comprise a node-set.

### 1.5.3 Document Order

The concept of document order might seem self-evident, but it is an important concept because the process and order by which nodes are evaluated depend on it. In essence, document order is the order of nodes as they are encountered while traversing the document as it would be read left-to-right and top-to-bottom.

The elements in our `<year>` example, in document order, would be: `<year>`, `<planting>`, the `<season>` with the period attribute with a value of spring, then the `<month>` elements in the first `<season>`—March, April, May—followed by the `<season>` with the period attribute with a value of summer and its `<month>` contents, and so on. In other words, document order is what you would expect as the order according to the direction, or sequence, in which the data is read.

Sometimes, a sequence of “reverse document order” can be stipulated, which means, as you would expect, the opposite of the order in which the content would be read at the node level. The reversing of document order occurs based on a starting node. If, in the example above, an expression referred to the second `<month>` starting from July, in reverse document order, it would be referring to the `<month>` node that contained the string June. It would not, for instance, be referring to "enuJ", or the string value being read in reverse. The first element in document order is the starting element, in this case, the `<month>` containing July, so the second element in reverse document order is the `<month>` containing June.

The document order of nodes is based on the tree hierarchy of the XML instance. The first node, then, would be the root node, or document root. Element nodes are ordered prior to their children, so the first element node would be the document element (`<year>` in our example), followed by its children. Nodes are selected in document order based on their starting tag,
or *opening tag*. Children nodes are processed prior to sibling nodes, and closing tags, or *end tags*, are implicitly ignored. Attribute and namespace nodes of a given element are ordered prior to the children of the element. This can be more readily seen in Figure 1–7, which shows the document order for the nodes from our `<year>` (with a few extras thrown in to demonstrate their position).

![Figure 1-7: Document order using six node types.](image)

1. The root node
2. The document element `<year>`
3. The attribute for the `iowa` namespace declaration
4. The element node `<planting>`
5. The element node `<season>`
6. The `iowa` namespace node
7. The `period` attribute with a value of "Spring"
8. The element node `<month>`
9. The March text node
10. The element node `<month>`
11. The April text node
12. The element node `<month>`
13. The May text node
14. The element node `<season>`
15. The iowa namespace node
16. The period attribute with a value of "Summer"
17. The element node `<month>`
18. The June text node
19. The element node `<month>`
20. The July text node
21. The element node `<month>`
22. The August text node
23. The comment node

Figure 1–7 displays the document order for six of the seven node types, the seventh being the processing-instruction node, which is not included in this example.

The only other possible order for the above nodes is the exact reverse, if specified in an expression as reverse document order. However, there is a mechanism in XSLT that will allow the sorting of nodes, which would then change the order of the nodes to something other than document order. Sorting will be addressed further in Chapter 9.

1.6 Climbing ‘Round the Family Tree: Addressing in XSLT

Navigation in XSLT and XPath involves addressing the various nodes with respect to their relationship with one another. If you put “tree” and “relationships” together, a logical inference is to use a tree analogy to model how members of a family are related. So, with XPath, the terminology for how one node is positioned in an XML document instance with respect to another is done in terms of family terminology, or a family tree.

A family tree traces one’s parents, grandparents, and other ancestors. XML uses the same familial terminology to describe the relationships between the nodes of a document.
Up to this point, we have used tree representations to show the structure of elements in an XML document. Now we are going to show nesting using a different form of representation, using boxes to show nesting. In Figure 1–8, consider the representation of XML and the full concept of element nesting as it is presented. In this way, the logical structure of a simple XML document instance can be represented.

This example shows the document element, <year>, which contains (as a boxed set) the <planting> and <harvest> elements, and so on. The same logical structure can be represented in the tried and true tree paradigm, as shown in Figure 1–9.

In any family tree, the oldest traceable ancestor is always at the top. In this case, <year> is the ancestor of all the other nodes in the instance. In addition, we can say that <year> is the parent of both <planting>
and `<harvest>`. Accordingly, then, `<planting>` and `<harvest>` are both `children` of `<year>`. If we asked for the parent of `<harvest>`, we would get `<year>`; for the parent of `<planting>`, we would also get `<year>`. This means that `<planting>` and `<harvest>` can also be called `siblings` to one another.

As we go further down the tree, additional features of familial relationships come into play. The various `<season>` nodes are all `descendants` of `<year>`, and `<year>` is also their shared, or common, ancestor.

In this kind of terminology that is so crucial for XPath, we do not say “grandparent” or “great-grandparent.” Any predecessor in the element hierarchy of the logical structure that is more than one node level removed is an ancestor.6

Lines of descendancy are kept carefully intact in XML and XPath, just as they usually are in the human realm. Where `<harvest>` and `<planting>` are siblings, so also are both of them parents to their own respective sets of `<season>` elements. The two `<season>` elements with attribute values for period of fall and winter, are both siblings because they share the same parent, `<harvest>`. Similarly, the `<season>` elements with the period attribute values spring and summer are also siblings sharing the same parent, `<planting>`.

The summer `<season>`, then, is not a sibling to the winter `<season>`. They do not share the same parent. In life, we might call these cousins, but neither XPath nor XML standards as a whole use this terminology.7 Of

---
6. This is correct with common practice, as a grandparent is an ancestor as well as a grandparent, and is more an ancestor than a parent with respect to the direct progenitor relationship.
7. This is partly because “cousin,” among other things, can have many different familial significations in English-speaking cultures and so becomes quite problematic as an intuitive referent of a relationship.
course, every `<season>` is at the same logical level in this document’s hierarchy, but they are not siblings.

As a definition of node relationships, each `<month>` within the fall `<season>` is:

- A sibling of the other `<month>` elements in the same `<season>`
- A child of its parent `<season period="fall">`
- A descendant of `<harvest>`
- A descendant of `<year>`
- A descendant of the document root
- The parent of the text node it contains

Another thing that might not be apparent at first glance, but is an unspoken assumption in working with the familial terminology in the tree structure, is that each of the families represented in the node branch are single-parent families. Call it a reflection of the post-modern world, but it is an important distinction of XML that no node can have more than one parent. Any node (except the root node) can have many ancestors, but only one parent.

Another point to mention is the relationship between attributes and elements. An element node such as `<season>` is called the parent of the attribute node with the attribute name `period`. However, the attribute `period` is not considered the child of `<season>`.

With this basic set of terms in mind, you can point to any part of a document based on its logical structure. This makes the value of markup and the logical structure it provides quite apparent, if, for example, the markup was in another language. In the case of our year example, providing the weather in the hemisphere from which that language arose was compatible, it would be possible to refer to the same period without knowing the specific word for that month. XPath relies heavily on this familial structure to navigate through a document when processing a particular XSLT function.