

Namespaces

No XML is an island. While you might find it useful to write documents that use a single markup vocabulary, (witness the baseball examples of Chapters 4 and 5) it's even more useful to mix and match tags from different XML applications. For example, you may want to include a `BIOGRAPHY` element in each `PLAYER` element. Since the biography consists basically of free-form, formatted text, it's convenient to write it in well-formed HTML without reinventing all the tags for paragraphs, line breaks, list items, bold elements, and so forth from scratch.

The problem, however, is that when mixing and matching tags from different XML applications, you're likely to find the same tag used for two different things. Is a `TITLE` the title of a page or the title of a book? Is an `ADDRESS` the mailing address of a company or the email address of a Webmaster? Namespaces disambiguate these instances by associating a URI with each tag set, and attaching a prefix to each element to indicate which tag set it belongs to. Thus, you can have both `BOOK:TITLE` and `HTML:TITLE` elements or `POSTAL:ADDRESS` and `HTML:ADDRESS` elements instead of just one kind of `TITLE` or `ADDRESS`. This chapter shows you how to use namespaces.

What Is a Namespace?

XML enables developers to create their own markup languages for their own projects. These languages can be shared with individuals working on similar projects all over the world. One specific example of this is XSL. XSL is itself an XML application for styling XML documents. The XSL transformation language must output arbitrary, well-formed XML, possibly including XSL itself. Thus, you need a clear-cut means of distinguishing between those XML elements that are XSL transformation instructions and output XML elements, *even if they have the same names!*



In This Chapter

What is a namespace?

Namespace syntax

Namespaces in DTDs



Namespaces are the solution. They allow each element and attribute in a document to be placed in a different namespace. The XML elements that comprise XSL transformation instructions are placed in the `http://www.w3.org/XSL/Transform/1.0` namespace. The XML elements that are part of the output can reside in some other convenient namespace like `http://www.w3.org/TR/REC-html40` or `http://www.w3.org/XSL/Format/1.0`. The exact namespace isn't even important as long as it's different.



Caution

If you're familiar with the concept of namespaces as used in C++ and other programming languages, you need to put aside your preconceptions before reading further. XML namespaces are similar to, but not quite the same as the namespaces used in programming. In particular, XML namespaces do not necessarily form a set (a collection with no duplicates).

Listing 15-2, a transformation from a source vocabulary to XSL formatting objects, initially appeared in Chapter 15, *XSL Formatting Objects*. It displays an XSL style sheet that converts from input XML to XSL formatting objects. The formatting engine distinguishes between elements that are XSL instructions and literal data for the output by using namespaces. Any element in the `http://www.w3.org/XSL/Transform/1.0` namespace represents a transformation instruction. Any element in the `http://www.w3.org/XSL/Format/1.0` namespace comprises part of the output.

```
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
  xmlns:fo="http://www.w3.org/XSL/Format/1.0"
  result-ns="fo" indent-result="yes">

  <xsl:template match="/">
    <fo:root xmlns:fo="http://www.w3.org/XSL/Format/1.0">

      <fo:layout-master-set>
        <fo:simple-page-master page-master-name="only">
          <fo:region-body/>
        </fo:simple-page-master>
      </fo:layout-master-set>

      <fo:page-sequence>

        <fo:sequence-specification>
          <fo:sequence-specifier-single page-master-name="only"/>
        </fo:sequence-specification>

        <fo:flow>
          <xsl:apply-templates select="//ATOM"/>
        </fo:flow>

      </fo:page-sequence>

    </fo:root>
  </xsl:template>
```

```
<xsl:template match="ATOM">
  <fo:block font-size="20pt" font-family="serif">
    <xsl:value-of select="NAME"/>
  </fo:block>
</xsl:template>

</xsl:stylesheet>
```

More specifically, these elements exist in the <http://www.w3.org/XSL/Transform/1.0> namespace and are XSL instructions:

- ♦ stylesheet
- ♦ template
- ♦ apply-templates
- ♦ value-of

These elements, in the <http://www.w3.org/XSL/Format/1.0> namespace, are XSL formatting objects and part of the output:

- ♦ root
- ♦ layout-master-set
- ♦ simple-page-master
- ♦ region-body
- ♦ sequence-specification
- ♦ sequence-specifier-single
- ♦ page-sequence
- ♦ block

The four elements with the `xsl` prefix have the *qualified names* beginning with the prefix:

- ♦ `xsl:stylesheet`
- ♦ `xsl:template`
- ♦ `xsl:apply-templates`
- ♦ `xsl:value-of`

However, their full names use the URL rather than the prefix:

- ♦ `http://www.w3.org/XSL/Transform/1.0:stylesheet`
- ♦ `http://www.w3.org/XSL/Transform/1.0:template`
- ♦ `http://www.w3.org/XSL/Transform/1.0:apply-templates`
- ♦ `http://www.w3.org/XSL/Transform/1.0:value-of`

In essence, the shorter qualified names are nicknames that are used only within the document because URLs often contain characters like ~, %, and / that aren't legal in XML names. However, qualified names do make documents a little easier to type and read.



Caution

Namespaces in an XML is an official W3C recommendation. The W3C considers it complete, aside from possible minor errors and elucidations. Nonetheless, of all the XML specifications from the W3C, this one is the most controversial. Many people feel very strongly that this standard contains fundamental flaws. The main objection argues that namespaces are, in practice, incompatible with DTDs and validation. While I don't have a strong opinion on this one way or the other, I do question the wisdom of publishing a standard when nothing approaching a consensus has been reached. Namespaces are a crucial part of many XML related specifications such as XSL and XHTML, so you need to understand them. Nonetheless, a lot of developers and authors have chosen to ignore this specification for their own work.

Namespace Syntax

Namespaces have been crafted to layer on top of the XML 1.0 specification. An XML 1.0 processor that knows nothing about namespaces can still read a document that uses namespaces, and will not find any errors. Documents that use namespaces do not break existing XML parsers (at least ones that don't check for validity); and users don't have to wait for notoriously unpunctual software companies to release expensive upgrades before using namespaces.

Definition of Namespaces

Namespaces are defined using an `xmlns:prefix` attribute on the applicable elements they apply to. *prefix* is replaced by the actual prefix used for the namespace. The value of the attribute is the URI of the namespace. For example, this `xsl:stylesheet` tag associates the prefix `xsl` with the URI `http://www.w3.org/XSL/Transform/1.0`.

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
```

The `xsl` prefix can then be attached to the local element and attribute names within the `xsl:stylesheet` element to identify them as belonging to the `http://www.w3.org/XSL/Transform/1.0` namespace. The prefix is separated from the local name by a colon. Listing 14-2, *XSL Transformations*, demonstrates by using the `xsl` prefix on the `stylesheet`, `template`, and `apply-templates` elements.

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

```

<xsl:template match="PERIODIC_TABLE">
  <html>
    <xsl:apply-templates/>
  </html>
</xsl:template>

<xsl:template match="ATOM">
  <P>
    <xsl:apply-templates/>
  </P>
</xsl:template>

</xsl:stylesheet>

```

The URI that defines a namespace is purely formal. Its only purpose is to group and disambiguate element and attribute names in the document. It does not necessarily point to anything. In particular, there is no guarantee that the document at the URI describes the syntax used in the document; or, for that matter, that any document exists at the URI. Having said that, if there is a canonical URI for a particular XML application, then that URI is a good choice for the namespace definition.

A namespace prefix can be any legal XML name that does not contain a colon. Recall from Chapter 6, *Well-Formed XML Documents*, that a legal XML name must begin with a letter or an underscore (_). Subsequent letters in the name may include letters, digits, underscores, hyphens, and periods. They may not include whitespace.


Note

There are two prefixes that are specifically disallowed, `xml` and `xmlns`. The `xml` prefix is defined to refer to <http://www.w3.org/XML/1998/namespace>. The `xmlns` prefix is used to bind elements to namespaces, and is therefore not available as a prefix to be bound to.

Other than disallowing the colon character in XML names (aside from its use in separating prefixes and local names) namespaces have no direct effect on standard XML syntax. A document that uses namespaces must still be well-formed when read by a processor that knows nothing about namespaces. If the document is to be validated, then it must be validated without specifically considering the namespaces. To an XML processor, a document that uses namespaces is just a funny-looking document in which some of the element and attribute names may have a single colon.


Caution

Namespaces do present problems for validation. If a DTD was written without namespace prefixes, then it must be rewritten using the namespace prefixes before it can be used to validate documents that use the prefixes. For example, consider this element declaration:

```
<!ELEMENT DIVISION (DIVISION_NAME, TEAM+)>
```

You have to rewrite it like this if the elements are all given the `bb` namespace prefix:

```
<!ELEMENT bb:DIVISION (bb:DIVISION_NAME, bb:TEAM+)>
```

This means that you cannot use the same DTD for both documents with namespaces and documents without, even if they use essentially the same vocabulary. In fact, you can't even use the same DTD for documents that use the same tag sets and namespaces, but different prefixes, because DTDs are tied to the actual prefixes rather than the URIs of the namespaces.

Multiple Namespaces

Listing 14-2 did not actually place the HTML elements in a namespace, but that's not hard to do. Listing 18-1 demonstrates. Just as `xsl` is the conventional prefix for XSL transformation instructions, `html` is the conventional prefix for HTML elements. In this example, the `xsl:stylesheet` element declares two different namespaces, one for XSL and one for HTML.

Listing 18-1: An XSL stylesheet that uses the `http://www.w3.org/TR/REC-html40` namespace for output

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
                xmlns:html="http://www.w3.org/TR/REC-html40">

  <xsl:template match="PERIODIC_TABLE">
    <html:html>
      <xsl:apply-templates/>
    </html:html>
  </xsl:template>

  <xsl:template match="ATOM">
    <html:p>
      <xsl:apply-templates/>
    </html:p>
  </xsl:template>

</xsl:stylesheet>
```

While it's customary, and generally useful to place the `xmlns` attribute on the root element, it can appear on other elements as well. In this case, the namespace prefix is understood only within the element where it's declared. Consider Listing 18-2. The `html` prefix is legal only in the `xsl:template` element where it's declared. It can't be applied in other template rules, unless they separately declare the `html` namespace.

Listing 18-2: An XSL style sheet with the `http://www.w3.org/TR/REC-html40` namespace declared in the template rules

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

  <xsl:template match="PERIODIC_TABLE"
                xmlns:html="http://www.w3.org/TR/REC-html40">
    <html:html>
      <xsl:apply-templates/>
    </html:html>
  </xsl:template>

  <xsl:template match="ATOM">
    <p>
      <xsl:apply-templates/>
    <p>
  </xsl:template>

</xsl:stylesheet>
```

You can redefine a namespace in a child element. For example, consider the XSL style sheet in Listing 18-3. Here, the `xsl` prefix appears in different elements to refer to `http://www.w3.org/XSL/Transform/1.0` and `http://www.w3.org/XSL/Format/1.0` alternately. Although every element has the prefix `xsl`, the XSL transformation instructions and the XSL formatting objects still reside in different namespaces because the meaning of the `xsl` prefix changes from element to element.

Listing 18-3: Redefining the `xsl` prefix

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

  <xsl:template match="/">
    <xsl:root xmlns:xsl="http://www.w3.org/XSL/Format/1.0">

      <xsl:layout-master-set>
        <xsl:simple-page-master page-master-name="only">
          <xsl:region-body/>
        </xsl:simple-page-master>
```

Continued

Listing 18-3 (continued)

```

</xsl:layout-master-set>

<xsl:page-sequence>

<xsl:sequence-specification>
  <xsl:sequence-specifier-single page-master-name="only"/>
</xsl:sequence-specification>

  <xsl:flow>
    <xsl:apply-templates select="//ATOM"/
      xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"/>
  </xsl:flow>

</xsl:page-sequence>

</xsl:root>
</xsl:template>

<xsl:template match="ATOM">
  <xsl:block font-size="20pt" font-family="serif"
    xmlns:xsl="http://www.w3.org/XSL/Format/1.0">
    <xsl:value-of select="NAME"
      xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"/>
  </xsl:block>
</xsl:template>

</xsl:stylesheet>

```

This is, however, needlessly confusing and I strongly recommend that you avoid it. There are more than enough prefixes to go around, and almost no need to reuse them within the same document. The main importance of this is if two different documents from different authors that happen to reuse a similar prefix are being combined. This is a good reason to avoid short prefixes like `a`, `m`, and `x` that are likely to be reused for different purposes.

Attributes

Since attributes belong to particular elements, they're more easily disambiguated from similarly named attributes without namespaces. Consequently, it's not nearly as essential to add namespaces to attributes as to elements. For example, the April 21, 1999 working draft of the XSL specification requires that all XSL transformation elements fall in the `http://www.w3.org/XSL/Transform/1.0` namespace. However, it does not require that the attributes of these elements be in any particular namespace. (In fact, it requires that they not be in any namespace.) Nonetheless, you can attach namespace prefixes to attributes if necessary. For example, this `PLAYER`

element and all its attributes live in the `http://metalab.unc.edu/xml/baseball` namespace.

```
<bb:PLAYER xmlns:bb="http://metalab.unc.edu/xml/baseball"
  bb:GIVEN_NAME="Tom" bb:SURNAME="Glavine"
  bb:POSITION="Starting Pitcher" bb:GAMES="33"
  bb:GAMES_STARTED="33" bb:WINS="20" bb:LOSSES="6" bb:SAVES="0"
  bb:COMPLETE_GAMES="4" bb:SHUT_OUTS="3" bb:ERA="2.47"
  bb:INNINGS="229.1" bb:HOME_RUNS_AGAINST="13"
  bb:RUNS_AGAINST="67" bb:EARNED_RUNS="63" bb:HIT_BATTER="2"
  bb:WILD_PITCHES="3" bb:BALK="0" bb:WALKED_BATTER="74"
  bb:STRUCK_OUT_BATTER="157" />
```

This might occasionally prove useful if you need to combine attributes from two different XML applications on the same element.

It is possible (though mostly pointless) to associate the same namespace URI with two different prefixes. There's really no reason to do this. The only reason I bring it up here is simply to warn you that it is the full name of the attribute that must satisfy XML's rules for an element not having more than one attribute with the same name. For example, this is illegal because `bb:GIVEN_NAME` and `baseball:GIVEN_NAME` are the same:

```
<bb:PLAYER xmlns:bb="http://metalab.unc.edu/xml"
  xmlns:baseball="http://metalab.unc.edu/xml"
  bb:GIVEN_NAME="Hank" bb:SURNAME="Aaron"
  baseball:GIVEN_NAME="Henry" />
```

On the other hand, the URI does not actually get checked to see what it points to. The URIs `http://metalab.unc.edu/xml/` and `http://www.metalab.unc.edu/xml/` point to the same page. However, this is legal:

```
<bb:PLAYER xmlns:bb="http://metalab.unc.edu/xml"
  xmlns:baseball="http://www.metalab.unc.edu/xml"
  bb:GIVEN_NAME="Hank" bb:SURNAME="Aaron"
  baseball:GIVEN_NAME="Henry" />
```

Default Namespaces

In long documents with a lot of markup, all in the same namespace, you might find it inconvenient to add a prefix to each element name. You can attach a default namespace to an element and its child elements using an `xmlns` attribute with no prefix. The element itself, as well as all its children, are considered to be in the defined namespace unless they possess an explicit prefix. For example, Listing 18-4 shows an XSL style sheet that does not prefix XSL transformation elements with `xsl` as is customary.



Note

Attributes are never in a default namespace. They must be explicitly prefixed.

Listing 18-4: An XSL stylesheet that uses default namespaces

```

<?xml version="1.0"?>
<stylesheet
  xmlns="http://www.w3.org/XSL/Transform/1.0"
  xmlns:fo="http://www.w3.org/XSL/Format/1.0"
  result-ns="fo">

  <template match="/">
    <fo:root xmlns:fo="http://www.w3.org/XSL/Format/1.0">

      <fo:layout-master-set>
        <fo:simple-page-master page-master-name="only">
          <fo:region-body/>
        </fo:simple-page-master>
      </fo:layout-master-set>

      <fo:page-sequence>

        <fo:sequence-specification>
          <fo:sequence-specifier-single page-master-name="only"/>
        </fo:sequence-specification>

        <fo:flow>
          <apply-templates select="//ATOM"/>
        </fo:flow>

      </fo:page-sequence>

    </fo:root>
  </template>

  <template match="ATOM">
    <fo:block font-size="20pt" font-family="serif">
      <value-of select="NAME"/>
    </fo:block>
  </template>

</stylesheet>

```

Perhaps the best use of default namespaces attaches a namespace to every element in an existing document to which you're now going to add tags from a different language. For instance, if you place some MathML in an HTML document, you only have to add prefixes to the MathML elements. You could put all the HTML elements in the <http://www.w3.org/TR/REC-html40> namespace simply by replacing the `<html>` start tag with this tag:

```
<html xmlns="http://www.w3.org/TR/REC-html40">
```

You do not need to edit the rest of the file! The MathML tags you insert still need to be in a separate namespace. However, as long as they aren't mixed up with a lot of HTML markup, you can simply declare an `xmlns` attribute on the root element of the MathML. This defines a default namespace for the MathML elements that override the default namespace of the document containing the MathML. Listing 18-5 demonstrates.

Listing 18-5: A MathML math element embedded in a well-formed HTML document that uses namespaces

```
<?xml version="1.0"?>
<html xmlns="http://www.w3.org/TR/REC-html40">
  <head>
    <title>Fiat Lux</title>
    <meta name="GENERATOR" content="amaya V1.3b" />
  </head>
  <body>

    <P>And God said,</P>

    <math xmlns="http://www.w3.org/TR/REC-MathML/">
      <mrow>
        <msub>
          <mi>&#x3B4;</mi>
          <mi>&#x3B1;</mi>
        </msub>
        <msup>
          <mi>F</mi>
          <mi>&#x3B1;&#x3B2;</mi>
        </msup>
        <mi></mi>
        <mo>=</mo>
        <mi></mi>
        <mfrac>
          <mrow>
            <mn>4</mn>
            <mi>&#x3C0;</mi>
          </mrow>
          <mi>c</mi>
        </mfrac>
        <mi></mi>
        <msup>
          <mi>J</mi>
          <mrow>
            <mi>&#x3B2;</mi>
            <mo></mo>
          </mrow>
        </msup>
      </math>
```

Continued

Listing 18-5 (continued)

```

        </mrow>
    </math>

    <P>and there was light</P>

</body>
</html>

```

Here, `math`, `mrow`, `msub`, `mo`, `mi`, `mfrac`, `mn`, and `msup` are all in the `http://www.w3.org/TR/REC-MathML/` namespace, even though the document that contains them uses the `http://www.w3.org/TR/REC-htm140` namespace.

Namespaces in DTDs

Namespaces do not get any special exemptions from the normal rules of well-formedness and validity. For a document that uses namespaces to be valid, the `xmlns` attributes must be declared in the DTD for those elements to which they're attached. Furthermore, you must declare the elements and attributes using the prefixes they use in the document. For instance, if a document uses a `math:subset` element, then the DTD must declare a `math:subset` element, not merely a `subset` element. (Of course, these rules do not apply to the merely well-formed documents discussed thus far.) For example:

```
<!ELEMENT math:subset EMPTY>
```

Default attribute values and `#IMPLIED` attributes can help here. For example, this `ATTLIST` declaration places every `math:subset` element in the `http://www.w3.org/TR/REC-MathML/` namespace unless specified otherwise in the document.

```
<!ATTLIST math:subset
    xmlns:math "http://www.w3.org/TR/REC-MathML/" #IMPLIED>
```

When working with valid documents, default namespaces prove especially useful since they don't require you to add prefixes to all the elements. Adding prefixes to elements from an XML application whose DTD doesn't use prefixes will break validity.

There are, however, clear limits to how far default namespaces will take you. In particular, they are not sufficient to differentiate between two elements that use an element name in incompatible ways. For example, if one DTD defines a `HEAD` as containing a `TITLE` and a `META` element, and another DTD defines a `HEAD` as containing `#PCDATA`, then you will have to use prefixes in the DTD and the document to distinguish the two different `HEAD` elements.

Two different development efforts are underway that may (or may not) eventually solve the problem of merging incompatible DTDs from different domains. XML schemas may provide a more robust replacement for DTDs. XML fragments may enable different documents to be combined with more distinction between which parts come from where. However, neither of these is even close to finished. Consequently, for now, merging incompatible DTDs will probably require you to rewrite the DTD and your documents to use prefixes.

**Tip**

If you have a question about whether a document that uses namespaces is well-formed or valid, forget everything you know about namespaces. Simply treat the document as a normal XML document that happens to have some element and attribute names that contain colons. The document is as well-formed and valid as it is when you don't consider namespaces.

Summary

This chapter explained how to work with namespaces. In particular, you learned:

- ♦ Namespaces distinguish between elements and attributes of the same name from different XML applications.
- ♦ Namespaces are declared by an `xmlns` attribute whose value is the URI of the namespace. The document referred to by this URI need not exist.
- ♦ The prefix associated with a namespace is the part of the name of the `xmlns` attribute that follows the colon; for example `xmlns:prefix`.
- ♦ Prefixes are attached to all element and attribute names that belong to the namespace identified by the prefix.
- ♦ If an `xmlns` attribute has no prefix, it establishes a default namespace for that element and its child elements (but not for any attributes).
- ♦ DTDs must be written in such a fashion that a processor that knows nothing about namespaces can still parse and validate the document.

The next chapter explores the Resource Description Framework, RDF, an XML application for encoding meta-data and information structures.



