

XLinks

XLL (eXtensible Linking Language) is divided into two parts, XLinks and XPointers. XLink, the XML Linking Language, defines how one document links to another document. XPointer, the XML Pointer Language, defines how individual parts of a document are addressed. XLinks point to a URI (in practice, a URL) that specifies a particular resource. This URL may include an XPointer part that more specifically identifies the desired part or section of the targeted resource or document. This chapter explores XLinks. The next chapter explores XPointers.

XLinks versus HTML Links

The Web conquered the more established gopher protocol for one main reason: It was possible to embed hypertext links in documents. These links could embed images or let the user to jump from inside one document to another document or another part of the same document. To the extent that XML is rendered into other formats such as HTML for viewing, the same syntax HTML uses for linking can be used in XML documents. Alternate syntaxes can be converted into HTML syntax using XSL, as you saw in several examples in Chapter 14.

However, HTML linking has limits. For one thing, URLs are mostly limited to pointing out a single document. More granularity than that, such as linking to the third sentence of the 17th paragraph in a document, requires you to manually insert named anchors in the targeted file. It can't be done without write access to the document to which you're linking.

Furthermore, HTML links don't maintain any sense of history or relations between documents. Although browsers may track the path you've followed through a series of documents, such tracking isn't very reliable. From inside the HTML, there's no way to know from where a reader came. Links are purely one way. The linking document knows to whom it's linking, but not vice versa.



In This Chapter

XLinks versus HTML links

Simple links

Extended links

Out-of-line links

Extended link groups

How to rename XLink attributes



XLL is a proposal for more powerful links between documents. It's designed especially for use with XML documents, but some parts can be used with HTML files as well. XLL achieves everything possible with HTML's URL-based hyperlinks and anchors. Beyond this, however, it supports multidirectional links where the links run in more than one direction. Any element can become a link, not just the `A` element. Links do not even have to be stored in the same file as the documents they link. Furthermore, the XPointer part (described in the next chapter) allows links to arbitrary positions in an XML document. These features make XLL more suitable not only for new uses, but for things that can be done only with considerable effort in HTML, such as cross-references, footnotes, end notes, interlinked data, and more.



Caution

I should warn you that at the time of this writing (spring, 1999), XLL is still undergoing significant development and modification. Although it is beginning to stabilize, some bits and pieces likely will change by the time you read this.

Furthermore, there are no general-purpose applications that support arbitrary XLinks. That's because XLinks have a much broader base of applicability than HTML links. XLinks are not just used for hypertext connections and embedding images in documents. They can be used by any custom application that needs to establish connections between documents and parts of documents, for any reason. Thus, even when XLinks are fully implemented in browsers they may not always be blue underlined text that you click to jump to another page. They can be that, but they can also be both more and less, depending on your needs.

Simple Links

In HTML, a link is defined with the `<A>` tag. However, just as XML is more flexible with tags that describe elements, it is more flexible with tags that refer to external resources. In XML, almost any tag can be a link. Elements that include links are called *linking elements*.

Linking elements are identified by an `xlink:form` attribute with either the value `simple` or `extended`. Furthermore, each linking element contains an `href` attribute whose value is the URI of the resource being linked to. For example, these are three linking elements:

```
<FOOTNOTE xlink:form="simple" href="footnote7.xml">7</FOOTNOTE>
<COMPOSER xlink:form="simple" inline="true"
  href="http://www.users.interport.net/~beand/">
  Beth Anderson
</COMPOSER>
<IMAGE xlink:form="simple" href="logo.gif"/>
```

Notice that the elements have semantic names that describe the content they contain rather than how the elements behave. The information that these elements are links is included in the attributes of the tags.

These three examples are simple XLinks. Simple XLinks are similar to standard HTML links and are likely to be supported by application software before the more complex (and more powerful) extended links, so I'll begin with them. Extended links are discussed in the next section.

In the FOOTNOTE example above, the link target attribute's name is href. Its value is the relative URL footnote7.xml. The protocol, host, and directory of this document are taken from the protocol, host, and directory of the document in which this link appears.

In the COMPOSER example above, the link target attribute's name is href. The value of the href attribute is the absolute URL http://www.users.interport.net/~beand/ In the third example above, which is IMAGE, the link target attribute's name is href. The value of the href attribute is the relative URL logo.gif. Again, the protocol, host, and directory of this document are taken from the protocol, host, and directory of the document in which this link appears.

If your document has a DTD, these attributes must be declared like any other. For example, DTD declarations of the FOOTNOTE, COMPOSER, and IMAGE elements might look like this:

```
<!ELEMENT FOOTNOTE (#PCDATA)>
<!ATTLIST FOOTNOTE
  xlink:form  CDATA    #FIXED "simple"
  href        CDATA    #REQUIRED
>
<!ELEMENT COMPOSER (#PCDATA)>
<!ATTLIST COMPOSER
  xlink:form  CDATA    #FIXED "simple"
  href        CDATA    #REQUIRED
>
<!ELEMENT IMAGE EMPTY>
<!ATTLIST IMAGE
  xlink:form  CDATA    #FIXED "simple"
  href        CDATA    #REQUIRED
>
```

With these declarations, the xlink:form attribute has a fixed value. Therefore it does not need to be included in the instances of the elements, which you may now write more compactly like this:

```
<FOOTNOTE href="footnote7.xml">7</FOOTNOTE>
<COMPOSER href="http://www.users.interport.net/~beand/">
  Beth Anderson
</COMPOSER>
<IMAGE href="logo.gif"/>
```

Making an element a link element doesn't impose any restriction on other attributes or contents of an element. A link element may contain arbitrary children or other attributes, always subject to the restrictions of the DTD, of course. For example, here's a more realistic declaration of the `IMAGE` element. Note that most of the attributes don't have anything to do with linking.

```
<!ELEMENT IMAGE EMPTY>
<!ATTLIST IMAGE
  xlink:form CDATA #FIXED "simple"
  href CDATA #REQUIRED
  ALT CDATA #REQUIRED
  HEIGHT CDATA #REQUIRED
  WIDTH CDATA #REQUIRED
>
```

Descriptions of the Local Resource

A linking element may contain optional `content-role` and `content-title` elements that provide extra information and further describe the purpose of the link inside the document in which it appears. For example:

```
<AUTHOR href="http://www.macfaq.com/personal.html"
  content-title="author of the page"
  content-role="whom to contact for questions about this page">
  Elliotte Rusty Harold
</AUTHOR>
```

The `content-role` and `content-title` attributes describe the local resource — that is, the contents of the link element, which is Elliotte Rusty Harold in this example. These attributes, however, do not describe the remote resource, which is the document at `http://www.macfaq.com/personal.html` in this example. Thus, this example says that Elliotte Rusty Harold has the title “author of the page” and the role “whom to contact for questions about this page.” This does not necessarily have any relation to the document that is found at `http://www.macfaq.com/personal.html`.

The `content-title` attribute is generally used by an application reading the XML to show a bit of extra information to the user, perhaps in the browser's status bar or via a tool tip, when the user moves the mouse over the linked element. However, the application is not required to show this information to the user. It simply may do so if it chooses.

The `content-role` attribute indicates the purpose of the linked element in the document. The `content-role` attribute is similar to a processing instruction in that it's intended to pass data to the application reading the XML. It has no real purpose as XML, though, and applications are free to ignore it.

Like all other attributes, `content-title` and `content-role` should be declared in the DTD for all elements to which they belong. For example, this is a reasonable declaration for the above `AUTHOR` element:

```

<!ELEMENT AUTHOR (#PCDATA)>
<!ATTLIST AUTHOR
    xlink:form      CDATA      #FIXED      "simple"
    href            CDATA      #REQUIRED
    content-title   CDATA      #IMPLIED
    content-role    CDATA      #IMPLIED
>

```

Descriptions of the Remote Resource

The link element may contain optional `role` and `title` attributes that describe the remote resource, that is, the document or other resource to which the link points.

For example:

```

<AUTHOR href="http://www.macfaq.com/personal.html"
    title="Elliotte Rusty Harold's personal home page"
    role="further information about the author of this page"
    content-title="author of the page"
    content-role="whom to contact for questions about this page">
    Elliotte Rusty Harold
</AUTHOR>

```

The `role` and `title` attributes describe the remote resource, not the local element. The remote resource in the above example is the document at `http://www.macfaq.com/personal.html`. Thus, the above example says that the page at `http://www.macfaq.com/personal.html` has the title “Elliotte Rusty Harold’s personal home page” and the role “further information about the author of this page.” It is not uncommon, though it’s not required, for the `title` to be the same as the contents of the `TITLE` element of the page to which you are linking.

The application reading the XML might use these two attributes to show extra information to the user. However, the application is not required to show this information to the user or do anything with it.

The `role` attribute indicates the purpose of the remote resource (the one to which it’s linked) in the linking document (the one from which it’s linked). For example, it might distinguish between footnotes, endnotes, and citations.

As with all other attributes, the `title` and `role` attributes should be declared in the DTD for all the elements to which they belong. For example, this is a reasonable declaration for the above author element:

```

<!ELEMENT AUTHOR (#PCDATA)>
<!ATTLIST AUTHOR
    xlink:form      CDATA      #FIXED "simple"
    href            CDATA      #REQUIRED
    content-title   CDATA      #IMPLIED
    content-role    CDATA      #IMPLIED
    title           CDATA      #IMPLIED
    role            CDATA      #IMPLIED
>

```

Link Behavior

Link elements can contain three more optional attributes that suggest to applications how the remote resource is associated with the current page. These are:

1. `show`
2. `actuate`
3. `behavior`

The `show` attribute suggests how the content should be displayed when the link is activated, for example, by opening a new window to hold the content. The `actuate` attribute suggests whether the link should be traversed automatically or whether a specific user request is required. The `behavior` attribute can provide detailed information to the application about exactly how the link is to be traversed, such as a time delay before the link is traversed. These are all application dependent, however, and applications are free to ignore the suggestions.

The `show` Attribute

The `show` attribute has three legal values: `replace`, `new`, and `embed`.

With a value of `replace` when the link is activated (generally by clicking on it, at least in GUI browsers), the target of the link replaces the current document in the same window. This is the default behavior of HTML links. For example:

```
<COMPOSER href="http://www.users.interport.net/~beand/"
          show="replace">
  Beth Anderson
</COMPOSER>
```

With a value of `new`, activating the link opens a new window in which the targeted resource is displayed. This is similar to the behavior of HTML links when the `target` attribute is set to `_blank`. For example:

```
<WEBSITE href="http://www.quackwatch.com/" show="new">
  Check this out, but don't leave our site completely!
</WEBSITE>
```



Caution

Readers do not expect a new window to open after clicking a link. They expect that when they click a link, the new page will load into the current window, unless they specifically ask for the link to open in a new window.

Some companies are so self-important that they find it impossible to believe that any user would ever want to leave their sites. Thus they “help” the readers by opening new windows. Most of the time this only serves to confuse and annoy. Don't change the behavior users expect without a very good reason. The thin hope that a reader might spend an additional two seconds on your site or view one more page and see one more ad is not a good reason.

With a value of `embed`, activating the link inserts the targeted resource into the existing document. Exactly what this means is application dependent. However, you can imagine it being used to provide client-side includes for Web pages. For example, this element, rather than directly including individual elements for the members of a family, copies them out of the separate files `ThomasCorwinAnderson.xml`, `LeAnahDeMintEnglish.xml`, `JohnJayAnderson.xml`, and `SamuelEnglishAnderson.xml`.

```
<FAMILY ID="f732">
  <HUSBAND href="ThomasCorwinAnderson.xml" show="embed"/>
  <WIFE href="LeAnahDeMintEnglish.xml" show="embed"/>
  <CHILD href="JohnJayAnderson.xml" show="embed"/>
  <CHILD href="SamuelEnglishAnderson.xml" show="embed"/>
</FAMILY>
```

The result, after the links are traversed and their contents embedded in the `FAMILY` element, is something like this:

```
<FAMILY ID="f732">
  <PERSON ID="p1035" SEX="M">
    <NAME>
      <GIVEN>Thomas Corwin</GIVEN>
      <SURNAME>Anderson</SURNAME>
    </NAME>
    <BIRTH>
      <DATE>24 Aug 1845</DATE>
    </BIRTH>
    <DEATH>
      <PLACE>Mt. Sterling, KY</PLACE>
      <DATE>18 Sep 1889</DATE>
    </DEATH>
  </PERSON>
  <PERSON ID="p1098" SEX="F">
    <NAME>
      <GIVEN>LeAnah (Lee Anna, Annie) DeMint</GIVEN>
      <SURNAME>English</SURNAME>
    </NAME>
    <BIRTH>
      <PLACE>Louisville, KY</PLACE>
      <DATE>1 Mar 1843</DATE>
    </BIRTH>
    <DEATH>
      <PLACE>acute Bright's disease, 504 E. Broadway</PLACE>
      <DATE>31 Oct 1898</DATE>
    </DEATH>
  </PERSON>
  <PERSON ID="p1102" SEX="M">
    <NAME>
      <GIVEN>John Jay (Robin Adair )</GIVEN>
      <SURNAME>Anderson</SURNAME>
    </NAME>
    <BIRTH>
```

```

        <PLACE>Sideview</PLACE>
        <DATE>13 May 1873</DATE>
    </BIRTH>
    <DEATH>
        <DATE>18 Sep 1889 </DATE>
    </DEATH>
</PERSON>
<PERSON ID="p37" SEX="M">
    <NAME>
        <GIVEN>Samuel English</GIVEN>
        <SURNAME>Anderson</SURNAME>
    </NAME>
    <BIRTH>
        <PLACE>Sideview</PLACE>
        <DATE>25 Aug 1871</DATE>
    </BIRTH>
    <DEATH>
        <PLACE>Mt. Sterling, KY</PLACE>
        <DATE>10 Nov 1919</DATE>
    </DEATH>
</PERSON>
</FAMILY>

```

Although each of these PERSON elements exists in a separate file, the complete FAMILY element is treated as though it was in one file.

Like all attributes in valid documents, the show attribute must be declared in a <!ATTLIST> declaration for the DTD's link element. For example:

```

<!ELEMENT WEBSITE (#PCDATA)>
<!ATTLIST WEBSITE
    xlink:form CDATA    #FIXED "simple"
    href        CDATA    #REQUIRED
    show        (new | replace | embed) "new"
>

```

The actuate Attribute

A link element's actuate attribute has two possible values: user and auto. The value user, the default, specifies that the link is to be traversed only when and if the user requests it. On the other hand, if the link element's actuate attribute is set to auto, the link is traversed any time one of the other targeted resources of the same link element is traversed. This is useful for link groups (discussed below).

Like all attributes in valid documents, the actuate attribute must be declared in the DTD in a <!ATTLIST> declaration for the link elements in which it appears. For example:

```

<!ELEMENT WEBSITE (#PCDATA)>
<!ATTLIST WEBSITE
    xlink:form CDATA    #FIXED "simple"
    href        CDATA    #REQUIRED

```

```

    show      (new | replace | embed) "new"
    actuate   (user | auto) "user"
>

```

The behavior Attribute

The `behavior` attribute is used to pass arbitrary data in an arbitrary format to the application reading the data. The application is expected to use this data to make additional determinations about how the link behaves. For example, if you want to specify that the sound file `fanfare.au` play when a link is traversed, you might write this:

```

<COMPOSER xlink:form="simple"
  href="http://www.users.interport.net/~beand/"
  behavior="sound: fanfare.au">
  Beth Anderson
</COMPOSER>

```

A Shortcut for the DTD

Because the attribute names and types are standardized, if you have more than one link element in a document, often it's convenient to make the attribute declarations a parameter entity reference and simply repeat that in the declaration of each linking element. For example:

```

<!ENTITY % link-attributes
  "xlink:form      CDATA      #FIXED 'simple'
  href            CDATA      #REQUIRED
  behavior        CDATA      #IMPLIED
  content-role    CDATA      #IMPLIED
  content-title   CDATA      #IMPLIED
  role            CDATA      #IMPLIED
  title           CDATA      #IMPLIED
  show            (new | replace | embed) 'new'
  actuate         (user | auto) 'user'
  behavior        CDATA      #IMPLIED"
>
<!ELEMENT COMPOSER (#PCDATA)>
<!ATTLIST COMPOSER
  %link-attributes;
>
<!ELEMENT AUTHOR (#PCDATA)>
<!ATTLIST AUTHOR
  %link-attributes;
>
<!ELEMENT WEBSITE (#PCDATA)>
<!ATTLIST WEBSITE
  %link-attributes;
>

```

However, this requires that the application reading the XML file understand that a `behavior` attribute with the value `sound: fanfare.au` means the sound file `fanfare.au` should play when the link is traversed. Most, probably all, applications don't understand this. However, they may use the `behavior` attribute as a convenient place to store nonstandard information they do understand.

As with all attributes in valid documents, the `behavior` attribute must be declared in the DTD for the link elements in which it appears. For example, the above `COMPOSER` element could be declared this way:

```
<!ELEMENT COMPOSER (#PCDATA)>
<!ATTLIST COMPOSER
    xlink:form CDATA    #FIXED "simple"
    href       CDATA    #REQUIRED
    behavior   CDATA    #IMPLIED
>
```

Extended Links

Simple links behave more or less like the standard links you're accustomed to from HTML. Each contains a single local resource and a reference to a single remote resource. The local resource is the link element's contents. The remote resource is the link's target.

Extended links, however, go substantially beyond what you can do with an HTML link to include multidirectional links between many documents and out-of-line links. Extended links are identified by an `xlink:form` attribute with the value `extended`, like this:

```
<WEBSITE xlink:form="extended">
```

The first capability of extended links is to point to more than one target. To allow this, extended links store the targets in child `locator` elements of the linking element rather than in a single `href` attribute of the linking element as simple links do. For example:

```
<WEBSITE xlink:form="extended">Cafe au Lait
  <locator href="http://metalab.unc.edu/javafaq/">
    North Carolina
  </locator>
  <locator
    href="http://sunsite.univie.ac.at/jcca/mirrors/javafaq/">
    Austria
  </locator>
  <locator href="http://sunsite.icm.edu.pl/java-corner/faq/">
    Poland
  </locator>
  <locator href="http://sunsite.uakom.sk/javafaq/">
```

```

    Slovakia
  </locator>
  <locator href="http://sunsite.cnlab-switch.ch/javafaq/">
    Switzerland
  </locator>
</WEBSITE>

```

Both the linking element itself, WEBSITE, in this example, and the individual locator children may have attributes. The linking element only has attributes that apply to the entire link and the local resource, such as content-title and content-role. The locator elements have attributes that apply to the particular remote resource to which they link, such as role and title. For example:

```

<WEBSITE xlink:form="extended" content-title="Cafe au Lait"
  content-role="Java news">
  <locator href="http://metalab.unc.edu/javafaq/"
    title="Cafe au Lait" role=".us"/>
  <locator
    href="http://sunsite.univie.ac.at/jcca/mirrors/javafaq/"
    title="Cafe au Lait" role=".at"/>
  <locator href="http://sunsite.icm.edu.pl/java-corner/faq/"
    title="Cafe au Lait" role=".pl"/>
  <locator href="http://sunsite.uakom.sk/javafaq/"
    title="Cafe au Lait" role=".sk"/>
  <locator href="http://sunsite.cnlab-switch.ch/javafaq/"
    title="Cafe au Lait" role=".ch"/>
</WEBSITE>

```

The actuate, behavior, and show attributes, if present, belong to the individual locator elements.

In some cases, as in the above example, where the individual locators point to mirror copies of the same page, remote resource attributes for individual locator elements may be the same across the linking element. In this case, you can use remote resource attributes in the linking element itself. These attributes apply to each of the locator children that does not declare a conflicting value for the same attribute. For example:

```

<WEBSITE xlink:form="extended" content-title="Cafe au Lait"
  content-role="Java news" title="Cafe au Lait">
  <locator href="http://metalab.unc.edu/javafaq/" role=".us"/>
  <locator
    href="http://sunsite.univie.ac.at/jcca/mirrors/javafaq/"
    role=".at"/>
  <locator href="http://sunsite.icm.edu.pl/java-corner/faq/"
    role=".pl"/>
  <locator href="http://sunsite.uakom.sk/javafaq/" role=".sk"/>
  <locator href="http://sunsite.cnlab-switch.ch/javafaq/"
    role=".ch"/>
</WEBSITE>

```

Another Shortcut for the DTD

If you have many `link` and `locator` elements, it may be advantageous to define the common attributes in parameter entities in the DTD, which you can reuse in different elements. For example:

```
<!ENTITY % remote-resource-semantic.att
  "role          CDATA          #IMPLIED
   title         CDATA          #IMPLIED
   show          (embed|replace|new) #IMPLIED 'replace'
   actuate       (auto|user)     #IMPLIED 'user'
   behavior      CDATA          #IMPLIED"
>

<!ENTITY % local-resource-semantic.att
  "content-title CDATA          #IMPLIED
   content-role  CDATA          #IMPLIED"
>

<!ENTITY % locator.att
  "href          CDATA          #REQUIRED"
>

<!ENTITY % link-semantic.att
  "inline       (true|false)    'true'
   role         CDATA          #IMPLIED"
>

<!ELEMENT WEBSITE (locator*) >
<!ATTLIST WEBSITE
  xlink:form CDATA #FIXED "extended"
  %local-resource-semantic.att;
>

<!ELEMENT locator EMPTY>
<!ATTLIST locator
  xlink:form CDATA #FIXED "locator"
  %locator.att;
  %link-semantic.att;
>
```

As always, in valid documents, the `link` elements and all their possible attributes must be declared in the DTD. For example, the following declares the `WEBSITE` and `locator` elements used in the above examples, as well as their attributes:

```

<!ELEMENT WEBSITE (locator*) >
<!ATTLIST WEBSITE
  xlink:form      CDATA      #FIXED  "extended"
  content-title   CDATA      #IMPLIED
  content-role    CDATA      #IMPLIED
  title           CDATA      #IMPLIED
>
<!ELEMENT locator EMPTY>
<!ATTLIST locator
  xlink:form      CDATA      #FIXED  "locator"
  href            CDATA      #REQUIRED
  role            CDATA      #IMPLIED
>

```

Out-of-Line Links

The links considered so far, both simple and extended, are inline links. Inline links, such as the familiar `A` element from HTML, use the contents of the link element as part of the document that contains the link. It is shown to the reader.

XLinks can also be out-of-line. An out-of-line link may not be present in any of the documents it connects. Instead, the links are stored in a separate linking document. For example, this might be useful to maintain a slide show where each slide requires next and previous links. By changing the order of the slides in the linking document, you can change the targets of the previous and next links on each page without having to edit the slides themselves.

To mark a link as out-of-line, provide an `inline` attribute with the value `false`. For example, the following simple, out-of-line link describes a Web site using an empty element. An empty element has no content; in the case of a link it has no local resource. Therefore, it should not have `content-role` or `content-title` attributes that describe the local resource. It may have, as in this example, `role` and `title` attributes that describe the remote resource.

```

<WEBSITE xlink:form="simple" inline="false"
  href="http://metalab.unc.edu/xml/"
  title = "Cafe con Leche" role="XML News"/>

```



Note

Because all the links you've seen until now were inline links, they implicitly had `inline` attributes with the value `true`, the default.

Simple out-of-line links, as in the above example, are relatively rare. Much more common and useful are out-of-line extended links, as shown below:

```

<WEBSITE xlink:form="extended" inline="false">
  <locator href="http://metalab.unc.edu/javafaq/" role=".us"/>

```

```

<locator
  href="http://sunsite.univie.ac.at/jcca/mirrors/javafaq/"
  role=".at"/>
<locator href="http://sunsite.icm.edu.pl/java-corner/faq/"
  role=".pl"/>
<locator href="http://sunsite.uakom.sk/javafaq/" role=".sk"/>
<locator href="http://sunsite.cnlab-switch.ch/javafaq/"
  role=".ch"/>
</WEBSITE>

```

Something such as this might be stored in a separate file on a Web server in a known location where browsers can find and query it to determine the nearest mirror of a page they're looking for. The out-of-line-ness, however, is that this element does not appear in the document from which the link is activated.

This expands the abstraction of style sheets into the linking domain. A style sheet is completely separate from the document it describes and yet provides rules that modify how the document is presented to the reader. A linking document containing out-of-line links is separated from the documents it connects, yet it provides the necessary links to the reader. This has several advantages, including keeping more presentation-oriented markup separate from the document and allowing the linking of read-only documents.



Caution

Style sheets are *much* farther along than out-of-line links. There currently is no general proposal for how you attach “link sheets” to XML documents, much less how you decide which individual elements in a document are associated with which links.

One obvious choice is to add an `<?xml-linksheet?>` processing instruction to a document's prolog to specify where the links are found. The link sheet itself could use something akin to XSL select patterns to map links to individual XML elements. The selectors could even become the value of the `locator` element's `role` attribute.

Extended Link Groups

An extended link group element contains a list of links that connect a particular group of documents. Each document in the group is targeted by means of an extended link document element. It is the application's responsibility to understand how to activate and understand the connections between the group members.



Caution

I feel compelled to note that application support for link groups is at best hypothetical at the time of this writing. Although I can show you how to write such links, their actual implementation and support likely is some time away. Some of the details remain to be defined and likely will be implemented in vendor-specific fashions, at least initially. Still, they hold the promise of enabling more sophisticated linking than can be achieved with HTML.

An Example

For example, I've put the notes for a Java course I teach on my Web site. Figure 16-1 shows the introductory page. This particular course consists of 13 classes, each of which contains between 30 and 60 individual pages of notes. A table of contents is then provided for each class. Each of the several hundred pages making up the entire site has links to the previous document, the next document, and the table of contents (Top link) for the week, as shown in Figure 16-2. Putting it all together, this amounts to more than a thousand interconnections among this set of documents.

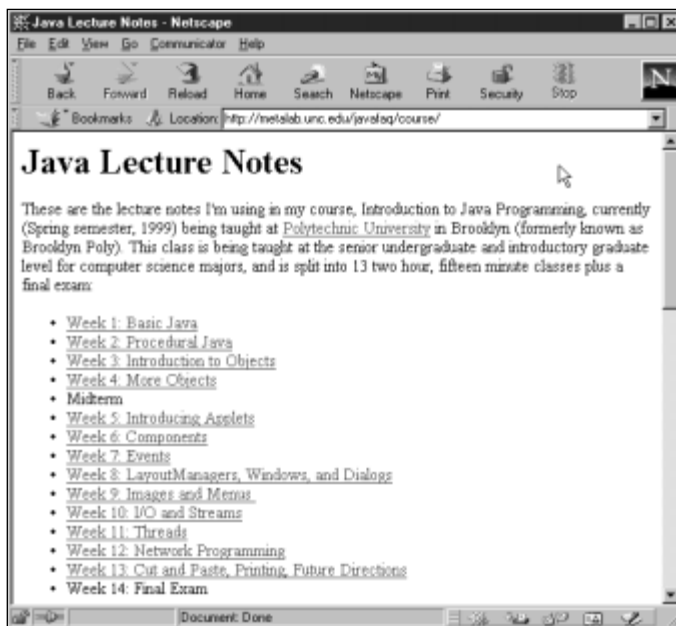


Figure 16-1: The introductory page for my class Web site shows 13 weeks of lecture notes

The possible interconnections grow exponentially with the number of documents. Every time a single document is moved, renamed, or divided into smaller pieces, the links need to be adjusted on that page, on the page before it and after it in the set, and on the table of contents for the week. Quite frankly, this is a lot more work than it should be, and it tends to discourage necessary modifications and updates to the course notes.

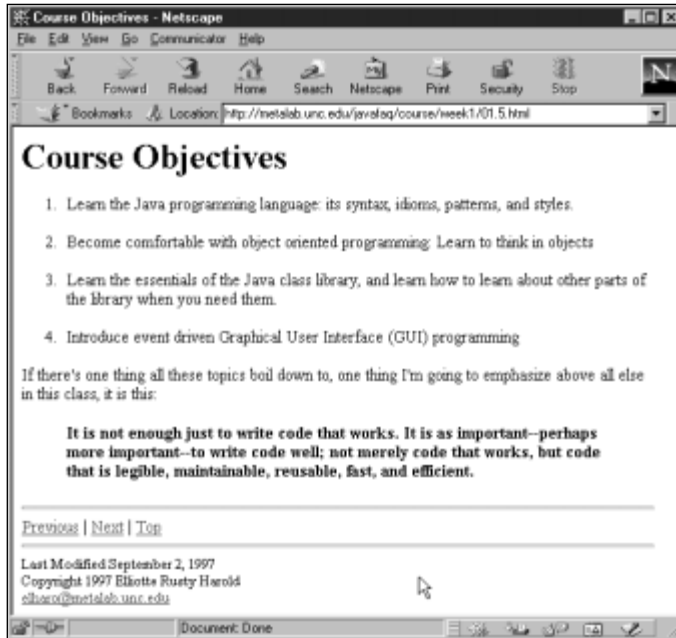


Figure 16-2: One page of lecture notes displaying the Previous, Next, and Top links

The sensible thing to do, if HTML supported it, would be to store the connections in a separate document. Reorganization of the pages then could be performed by editing that one document. HTML links don't support this, but XLinks do. Instead of storing the links inline in HTML files, they can be stored out-of-line in group elements. For example:

```
<COURSE xlink:form="group">
  <CLASS xlink:form="document" href="week1/index.xml"/>
  <CLASS xlink:form="document" href="week2/index.xml"/>
  <CLASS xlink:form="document" href="week3/index.xml"/>
  <CLASS xlink:form="document" href="week4/index.xml"/>
  <CLASS xlink:form="document" href="week5/index.xml"/>
  <CLASS xlink:form="document" href="week6/index.xml"/>
  <CLASS xlink:form="document" href="week7/index.xml"/>
  <CLASS xlink:form="document" href="week8/index.xml"/>
  <CLASS xlink:form="document" href="week9/index.xml"/>
  <CLASS xlink:form="document" href="week10/index.xml"/>
  <CLASS xlink:form="document" href="week11/index.xml"/>
  <CLASS xlink:form="document" href="week12/index.xml"/>
  <CLASS xlink:form="document" href="week13/index.xml"/>
</COURSE>
```

This defines the `COURSE` element as an extended link group, which consists of 13 extended link document elements, the `CLASS` elements.

The steps Attribute

One thing an application may choose to do with a link group is preload all the documents in the link group. These documents may contain link groups of their own. For example, each of the CLASS elements above refers to one of the site's table of contents pages for a specific week, as shown in Figure 16-3. These documents could then load. For example, the file week6/index.xml could contain this link group:

```
<CLASS xlink:form="group">
  <SLIDE xlink:form="document" href="01.xml"/>
  <SLIDE xlink:form="document" href="02.html"/>
  <SLIDE xlink:form="document" href="06.html"/>
  <SLIDE xlink:form="document" href="12.html"/>
  <SLIDE xlink:form="document" href="13.html"/>
  <SLIDE xlink:form="document" href="16.html"/>
  <SLIDE xlink:form="document" href="17.html"/>
  <SLIDE xlink:form="document" href="19.html"/>
  <SLIDE xlink:form="document" href="21.html"/>
  <SLIDE xlink:form="document" href="22.html"/>
  <SLIDE xlink:form="document" href="24.html"/>
</CLASS >
```



Figure 16-3: A table-of-contents page showing the first week's lecture notes

Now suppose one of these documents refers back to the original document. This might trigger an infinite regression, with the same documents repeatedly loading until the application runs out of memory. To prevent this, the group element may contain a `steps` attribute that specifies the number of levels to recursively follow link groups. For example, to specify that preloading shouldn't go deeper than three levels from the current document, write:

```
<group xlink:form="group" steps="3">
```


Note

To be honest, I'm not sure how important this is. It's not hard for an application to note when it's already followed a document and not process the document a second time. I suspect it is better to place the requirement for preventing recursion with the XML processor rather than the page author.

The `steps` attribute can be used to limit the amount of preloading that occurs. For instance, in the class notes example, it's unlikely that any person is going to read the entire set of course notes in one sitting, though perhaps he or she may want to print or copy all of them. In any case, by setting the `steps` attribute to 1, you can limit the depth of the traversal to simply the named pages rather than the several hundred pages in the course.

As always, these elements and their attributes must be declared in the DTD of any valid document in which they appear. In practice, the `xlink:form` attribute is fixed so that it need not be included in instances of the element. For example:

```
<!ELEMENT CLASS (document*)>
<!ATTLIST CLASS
  xlink:form CDATA #FIXED "group"
  steps      CDATA #IMPLIED
>
<!ELEMENT SLIDE EMPTY>
<!ATTLIST SLIDE
  xlink:form CDATA #FIXED "document"
  href       CDATA #REQUIRED
>
```

Renaming XLink Attributes

XLinks are built around the ten attributes discussed in the previous sections. They are listed below.

```
xlink:form
href
steps
title
role
content-title
content-role
show
```

actuate
behavior

It is far from inconceivable that one or more of these attributes will already be used as an attribute name in a particular XML application. The `title` attribute seems particularly likely to be taken. The only one that really shouldn't be used for other purposes is `xlink:form`.

The XLink specification anticipates this problem and allows you to rename the XLink attributes to something more convenient using the `xml:attributes` attribute. This attribute is declared in an `<!ATTLIST>` declaration in the DTD as a fixed attribute with type `CDATA` and a value that's a whitespace-separated list of pairs of standard names and new names.

Note

This is exactly the problem that namespaces (discussed in Chapter 18) were designed to solve. I would not be surprised to see this entire mechanism deleted in a future draft of XLL and replaced with a simple namespace prefix such as `xlink:.`

For example, the link elements shown in this chapter look a little funny because the standard names are all lowercase while this book's convention is all uppercase. It's easy enough to change the XLink attributes to uppercase with a declaration such as this:

```
<!ELEMENT WEBSITE (#PCDATA)>
<!ATTLIST WEBSITE
  xlink:form CDATA    #FIXED "simple"
  xml:attributes CDATA #FIXED
    "href HREF show SHOW actuate ACTUATE"
  HREF      CDATA    #REQUIRED
  SHOW      CDATA    (new | replace | embed) "new"
  ACTUATE   CDATA    (user | auto) user
>
```

Now you can rewrite the WEBSITE example in this more congruous form:

```
<WEBSITE HREF="http://www.microsoft.com/" SHOW="new">
  Check this out, but don't leave our site completely!
</WEBSITE>
```

The above `ATTLIST` declaration only changes the attributes of the WEBSITE element. If you want to change them the same way in multiple other examples, the easiest approach is to use a parameter entity:

```
<!ENTITY LINK_ATTS
  'xlink:form CDATA    #FIXED "simple"
  xml:attributes CDATA #FIXED
    "href HREF show SHOW actuate ACTUATE"
  HREF      CDATA    #REQUIRED
  SHOW      CDATA    (new | replace | embed) "new"
  ACTUATE   CDATA    (user | auto) "user"'
```

```

>
<!ELEMENT WEBSITE (#PCDATA)>
<!ATTLIST WEBSITE %LINK_ATTTS;>

<!ELEMENT COMPOSER (#PCDATA)>
<!ATTLIST COMPOSER %LINK_ATTTS;>

<!ELEMENT FOOTNOTE (#PCDATA)>
<!ATTLIST FOOTNOTE %LINK_ATTTS;>

```

Summary

In this chapter, you learned about XLinks. In particular you learned:

- ♦ XLinks can do everything HTML links can do and quite a bit more, but they aren't supported by current applications.
- ♦ Simple links behave much like HTML links, but they are not restricted to a single `<A>` tag.
- ♦ Link elements are identified by `xlink:form` and `href` attributes.
- ♦ Link elements can describe the local resource with `content-title` and `content-role` attributes.
- ♦ Link elements can describe the remote resource they're linking to with `title` and `role` attributes.
- ♦ Link elements can use the `show` attribute to tell the application how the content should be displayed when the link is activated, for example, by opening a new window.
- ♦ Link elements can use the `behavior` attribute to provide the application with detailed, application dependent information about exactly how the link is to be traversed.
- ♦ Link elements can use the `actuate` attribute to tell the application whether the link should be traversed without a specific user request.
- ♦ Extended links can include more than a single URI in a linking element. Currently, it's left to the application to decide how to choose between different alternatives.
- ♦ An extended link group element contains a list of links that connect a particular group of documents.
- ♦ You can use the `xml:attributes` attribute in the DTD to rename the standard XLink attributes such as `href` and `title`.

In the next chapter you see how XPointers can be used to link not only to remote documents, but to very specific elements in remote documents.

