XML Linking and Style

W3C NOTE 5 June 2001

This version:
http://www.w3.org/TR/2001/NOTE-xml-link-style-20010605/ (HTML, XML)
Latest version:
http://www.w3.org/TR/xml-link-style/
Editor:
Norman Walsh, Sun Microsystems, Inc. <Norman.Walsh@East.Sun.COM>

Abstract

The interaction of XLink linking elements and styling has not previously been carefully described. This note, the result of an XML Linking/XSL joint task force, attempts to rectify that oversight by providing a clear conceptual model for linking and styling and suggestions for the practical application of that model using current W3C Recommendations (and Working Drafts, Candidate Recommendations, and Proposed Recommendations).

Status of this Document

This document is a Note made available by the World Wide Web Consortium (W3C) for information only. Publication of this Note by W3C does not imply endorsement by W3C or the W3C Team, or any W3C Members, nor that the W3C has, is, or will be allocating any resources to the issues addressed by this Note.

It was produced by a joint task force of the XML Linking and XSL Working Groups. We plan to update this Note according to comments received and, if there is enough interest, consider promoting parts of it into Recommendation-track specifications where appropriate. Comments may be directed to the task force mailing list w3c-xsl-link-tf@w3.org (Member-only archive).

A list of current W3C Recommendations and other technical documents, including Working Drafts and Notes, can be found at http://www.w3.org/TR/.

This section represents the status of this document at the time this version was published. It will become outdated if and when a new version is published. The

latest status is maintained at the W3C.

Table of Contents

Appendices

---

1 Introduction

Linking and styling have significant interactions. On the one hand, style may be applied to elements because they participate in links. On the other, selecting a link may modify, replace, or create a new document which must then be styled.

This note introduces a conceptual model for describing the interactions of XLink linking elements and styling. It then shows how this model may be applied in two different ways:

1. Using current and anticipated technologies supported by existing W3C Recommendations (and Working Drafts, Candidate Recommendations, and Proposed Recommendations).
2. In an environment where the XSLT processor provides significantly more functionality for linking and contains several new features.

The full generality of the conceptual model cannot be represented with current Recommendations. Rather than attempt to restrict the model, we observe that by accepting default values for parts of the model, a useful, practical subset can be implemented today.

1.1 Design Principles

The task force used the following principles to develop the model described in this note:

1. The model must support applications of linking and styling for the purpose of presentation to a human audience.

   There are countless XML linking applications that are not related to style at all (robots, indexing applications, etc.). At best, the conceptual framework described in this note may aid in developing interoperable, standard applications of this sort. At the very least, nothing in this note should inhibit the development of XML linking applications that are not related to style.
2. The model must not compromise accessibility.
3. This model must apply equally well to XSL and CSS.
4. The consequences of linking to or from non-XML resources is outside the scope of this note.

1.2 Requirements Framework

In considering this design, we felt the following requirements had to be met by any

proposed solution:

1. The overall treatment of the ending resource is controlled by the `xlink:show` attribute.
2. The overall treatment of link traversal is controlled by the `xlink:actuate` attribute.
3. It must be possible to style both starting and ending resources.
4. It must be possible to process inbound, outbound, and third party arcs.
5. It must be possible to process starting resources regardless of "well-balancedness": whole documents, one or more whole nodes, an arbitrary range, including partial nodes, and non-contiguous selections of all of the above.
6. It must be possible to process ending resources regardless of "well-balancedness".
7. It must be possible to control the context in which ending resources are displayed. While whole documents are the standard context for web browsers, it is easy to imagine situations where it would be desirable to show a portion of the document that contains the target, or even just the target itself.
8. We must consider the consequences of encountering different sequences and nestings of starting resources with various combinations of behavior settings.

2 Processing XLink Linking Elements

Processing XLink linking elements can be divided into two functions: link identification and link traversal. It is useful for our purposes to consider these functions separately.

2.1 XML Link Identification

XLink linking elements may or may not participate in the link they identify. Consider the following example:

```
<doc>
<p><name xlink:type='simple'
      xlink:href='students/patjones62.xml'>Pat Jones</name>
is taking <course xlink:type='simple'
      xlink:href='#CS101'>CS-101</course>.</p>
<courselist>
<course id='CS101'>Computer Science 101</course>
<course id='CS201'>Computer Science 201</course>
</courselist>
</doc>
```

In this example, the two green (and underlined) elements are starting resources. The blue (and italic) element is an ending resource. There is no explicit markup to identify ending resources, but an XLink-aware application can identify them because they are pointed to by URI references.

Linking relationships can be less obvious. Consider the case where an extended

link is used:

```
<doc>
<extendedlink xlink:type='extended'>
  <loc xlink:type='locator' xlink:label='cs101'
      xlink:href='#xpointer(//course[@id='CS101'])'></loc>
  <loc xlink:type='locator' xlink:label='cs201'
      xlink:href='#xpointer(//course[@id='CS201'])'></loc>
  <loc xlink:type='locator' xlink:label='cs101desc'
      xlink:href='courses/cs101.xml'></loc>
  <loc xlink:type='locator' xlink:label='cs201desc'
      xlink:href='courses/cs201.xml'></loc>
  <arc xlink:type='arc' xlink:from='cs101' xlink:to='cs101desc'></arc>
  <arc xlink:type='arc' xlink:from='cs201' xlink:to='cs201desc'></arc>
</extendedlink>
<p><name xlink:type='simple'
      xlink:href='students/patjones62.xml'>Pat Jones</name>
is taking <course xlink:type='simple'
      xlink:href='#CS101'>CS-101</course>.</p>
<courselist>
<course id='CS101'>Computer Science 101</course>
<course id='CS201'>Computer Science 201</course>
</courselist>
</doc>
```

With the addition of an extended link element, the courses in the course list also become starting resources. Note that this makes the first course in the course list (identified in red and bold-italic) both a starting resource and an ending resource. This example could be complicated further by placing the extended link in a physically separate resource from the documents in question.

In order for links to be styled, they must be identified. The XLink-aware application must somehow provide additional information beyond what is explicitly marked up in the document available to the styling engine. This can be described in terms of Infoset contributions.

2.1.1 XLink Infoset Contributions

Conceptually, the XLink-aware application must provide additional information to the styling engine about links in a document. This can be modeled in terms of contributions to the document's Infoset.

> **Note:**
>
> We use the Infoset here as a conceptual model. We are not suggesting that applications be required to build or maintain any specific set of data structures for this purpose, merely that they be able to provide certain information to the application.
>
> Because it is convenient to describe some aspects of link processing in terms of the XPath data model, we use mostly Element Information Item in our contributions, but that does mean that implementations are required to construct their internal data models that way.

An XLink-aware processor has at its disposal some collection of XLinks. They may come from simple and/or extended links in the source document, or from external

linkbases. We treat them all equally.

To augment the Infoset, we introduce a new data structure, the Link Set, and three new properties to access it. The Link Set holds an interpretation of the XLink data model: the links, arcs, and participants that are known as well as any additional metadata about them that may be available.

We construct the Link Set using Element Information Items. Note, however, that these elements are not the children of any information items in the document's Infoset. They are accessible only through new properties. We elected to use Element Information Items for the Link Set because:

1. It is a hierarchical data structure with a natural tree structure.
2. Style engines already have well-understood mechanisms for navigating through a tree of elements and attributes.
3. XLinks can have additional metadata elements inside them, and it is possible for these elements to have semantic meaning, it is therefore necessary to provide a means for XLink processors to access them.

There are three types of Element Information Items in the Link Set: Link Information Items, which represent links, Arc Information Items which represent single arcs, and two Participant Information Items which represent the start and end of an arc.

More formally:

- [Definition: There is one **Link Information Item** for every XLink that the processor knows about.]
- [Definition: Every Link Information Item contains an **Arc Information Item** for every arc associated with that XLink.]

  In the case of simple links, there will only ever be a single arc, but extended links may include many. Note that the number of Arc Information Items is not necessarily the same as the number of `arc`-type elements in the XLink. XLink `arc`-type elements can define any number of arcs; Arc Information Items define exactly one arc.
- Finally, [Definition: Every Arc Information Item contains exactly two **Participant Information Items**: one Start Participant Information Item for the starting resource and one End Participant Information Item for the ending resource.]

Once constructed, the new information items must be inserted into the Infoset. We define new properties for this purpose.

- [Definition: The Document Information Item is extended with a new Link Set property. The **Link Set** property contains the set of Link Information Items associated with this document.]

Every Element, Character, Processing Instruction, and Comment Information Item in the document is augmented with two new properties: Starting Participant Arcs and Ending Participant Arcs.

- [Definition: The **Starting Participant Arcs** property is a list of the Arc Information Items for which this information item is a part of the Start Participant Information Item for the arc.] In other words, if this element is part of the starting resource of a link, it has a pointer to the arc(s) that define the traversal from this resource.
- [Definition: The **Ending Participant Arcs** property is a list of the Arc Information Items for which this information item is a part of the End Participant Information Item for for the arc.] Again, in other words, if this element is part of the ending resource of a link, it has a pointer to the arc(s) that define the traversal to this resource.

If an information item has more than one Arc Information Item associated with it, they are ordered. That is, one can refer to the first Arc Information Item, the second Arc Information Item, etc. There is no requirement for different implementations to select the same order, or even for the same implementation to select the same order over multiple runs. They need only be consistent for the logical duration of any given information set.

If more than one information item in the document is associated with an arc (for example, if the starting or ending resources are comprised of several elements or a range of characters), the Starting Participant Arcs or Ending Participant Arcs of each relevant information item in the Infoset is augmented.

Once the Infoset has been augmented with these items, linking applications can process links in a uniform way, independent of whether the elements in question are simple, extended, or even third party links.

> **Note:**
>
> We observe that this interface extends even beyond XLink. Processors for schemas that have linking semantics defined independent of XLink, such as HTML, can augment the Infoset and take advantage of this linking framework without literally making every link an XLink.

An XLink can be quite complex. In the most general case, linking semantics may be influenced by the roles of the `link`-type element, the `arc`-type element, and the roles on the participating resources involved in the link. In designing this model, our goal was to provide link processors with access to all of this information.

2.1.1.1 Link Element Namespace

In order to provide fully qualified names for the linking information items, they are placed in the linking and style namespace with Namespace name "http://www.w3.org/2001/06/xml-link-style".

This namespace name (URI) will only be used to refer to this version of this Note: different URIs will be used for any and all new versions of the specification except that this namespace name may be reused in any update of this Note which is made for the purpose of clarification or bug fixes. These changes will be minor in that they will not (a) change the meaning or validity of existing documents written using the namespace, or (b) affect the operation of existing software written to

process such documents.

2.1.1.2 Link Information Item

A Link Information Item is an Element Information Item. It must conform to the following [XML Schema (Structures)] type definition:

```
<xsd:schema targetNamespace="http://www.w3.org/2001/06/xml-link-style"
            elementFormDefault="qualified"
            xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
            xmlns:xlink="http://www.w3.org/1999/xlink"
            xmlns:link='http://www.w3.org/2001/06/xml-link-style'>
<xsd:complexType name='linkType'>
  <xsd:sequence minOccurs='1' maxOccurs='1'>
    <xsd:element ref='link:arc' minOccurs='1' maxOccurs='unbounded'/>
    <xsd:choice minOccurs='0' maxOccurs='unbounded'>
      <xsd:element name='xlink:title' type='opaqueType'/>
      <xsd:any namespace='##other' minOccurs='0' maxOccurs='unbounded'/
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name='type'>
    <xsd:simpleType>
      <xsd:restriction base='xsd:string'>
        <xsd:enumeration value='simple'/>
        <xsd:enumeration value='extended'/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name='role' type='xsd:uriReference'/>
  <xsd:attribute name='title' type='xsd:string'/>
  <xsd:anyAttribute namespace='##other' processContents='strict'/>
</xsd:complexType>
<xsd:element name='link' type='link:linkType'/>
</xsd:schema>
```

It has the following properties:

type

The value of the `xlink:type` attribute for this link: either "simple" or "extended".

role

The normalized value of the `xlink:role` attribute for this link.

This property is null if the `extended`-type element does not have an `xlink:role` attribute value or if the link is of the `simple` type.

title

The normalized value of the `xlink:title` attribute for this link.

This property is null if the `extended`-type element does not have an `xlink:title` attribute value or if the link is of the `simple` type.

##other:*

> It may have any other properties as long as they are from a different namespace.

A Link Information Item must contain at least one Arc Information Item and may contain any number of `xlink:title` elements and elements from other namespaces.

2.1.1.3 Arc Information Item

An Arc Information Item is an Element Information Item. It must conform to the following XML Schema type definition:

```
<xsd:schema targetNamespace="http://www.w3.org/2001/06/xml-link-style"
            elementFormDefault="qualified"
            xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
            xmlns:xlink="http://www.w3.org/1999/xlink"
            xmlns:link='http://www.w3.org/2001/06/xml-link-style'>
<xsd:complexType name='arcType'>
  <xsd:sequence minOccurs='1' maxOccurs='1'>
    <xsd:element ref='link:startParticipant'/>
    <xsd:element ref='link:endParticipant'/>
    <xsd:choice minOccurs='0' maxOccurs='unbounded'>
      <xsd:element name='xlink:title' type='opaqueType'/>
      <xsd:any namespace='##other' minOccurs='0' maxOccurs='unbounded'/
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name='role' type='xsd:uriReference'/>
  <xsd:attribute name='title' type='xsd:string'/>
  <xsd:attribute name='show'>
    <xsd:simpleType>
      <xsd:restriction base='xsd:string'>
        <xsd:enumeration value='new'/>
        <xsd:enumeration value='replace'/>
        <xsd:enumeration value='embed'/>
        <xsd:enumeration value='other'/>
        <xsd:enumeration value='none'/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name='actuate'>
    <xsd:simpleType>
      <xsd:restriction base='xsd:string'>
        <xsd:enumeration value='onLoad'/>
        <xsd:enumeration value='onRequest'/>
        <xsd:enumeration value='other'/>
        <xsd:enumeration value='none'/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name='target-stylesheet' type='xsd:uriReference'/>
  <xsd:attribute name='target-processing-context' type='xsd:uriReferenc
  <xsd:attribute name='target-presentation-context' type='xsd:uriRefere
  <xsd:anyAttribute namespace='##other' processContents='strict'/>
</xsd:complexType>
</xsd:schema>
```

It has the following properties:

role

The normalized value of the `xlink:arcrole` attribute for this arc.

This property is null if the relevant `arc`- or `simple`-type element does not have an `xlink:arcrole` attribute value.

title

The normalized value of the `xlink:title` attribute for this arc.

This property is null if the `extended`-type element does not have an `xlink:title` attribute value or if the link is of the `simple` type.

show

The normalized value of the `xlink:show` attribute for the arc for which this is the starting resource.

This property is null if the relevant `arc`- or `simple`-type element does not have an `xlink:show` attribute value.

actuate

The normalized value of the `xlink:actuate` attribute for the arc for which this is the starting resource.

This property is null if the relevant `arc`- or `simple`-type element does not have an `xlink:actuate` attribute value.

target-stylesheet

The stylesheet that should be used when this arc is traversed. (See **4 Styling Link Traversal**)

target-processing-context

A URI reference that identifies the processing context associated with traversal of this arc. (See **4 Styling Link Traversal**)

target-presentation-context

A URI reference that identifies the presentation context associated with traversal of this arc. (See **4 Styling Link Traversal**)

##other:*

It may have any other attributes as long as they are from a different namespace.

An Arc Information Item must contain a Start Participant Information Item and an End Participant Information Item. It may also contain any number of other elements from other namespaces.

2.1.1.4 Participant Information Item

A Participant Information Item is an Element Information Item. It must conform to the following XML Schema type definition:

```
<xsd:schema targetNamespace="http://www.w3.org/2001/06/xml-link-style"
            elementFormDefault="qualified"
            xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
            xmlns:xlink="http://www.w3.org/1999/xlink"
            xmlns:link='http://www.w3.org/2001/06/xml-link-style'>
<xsd:complexType name='participantType'>
  <xsd:choice minOccurs='0' maxOccurs='unbounded'>
    <xsd:element name='xlink:title' type='opaqueType'/>
    <xsd:any namespace='##other' minOccurs='0' maxOccurs='unbounded'/>
  </xsd:choice>
  <xsd:attribute name='role' type='xsd:uriReference'/>
  <xsd:attribute name='title' type='xsd:string'/>
  <xsd:attribute name='resource' type='xsd:string'/>
  <xsd:anyAttribute namespace='##other' processContents='strict'/>
</xsd:complexType>
</xsd:schema>
```

It has the following properties:

role

The normalized value of the `xlink:role` attribute for the `locator`- or `resource`-type element that specifies this starting resource.

This property is null if the relevant `locator`- or `resource`-type element does not have an `xlink:role` attribute value or if the link is of the `simple` type.

title

The normalized value of the `xlink:title` attribute for this participant.

This property is null if the relevant `locator`- or `resource`-type element does not have an `xlink:title` attribute value or if the link is of the `simple` type.

resource

A reference to the resource this participant identifies. We intentionally do not stipulate the type of this reference. Often it will be a URI reference (obtained from the `xlink:href` attribute on the relevant `locator`- or `simple`-type element), but in cases where there is no extant URI (for example, to identify the starting participant of a `simple`-type link), we do not wish to require that a URI reference be manufactured.

##other:*

> It may have any other properties as long as they are from a different namespace.

A Participant Information Item may contain any number of other elements from other namespaces.

2.1.1.5 Infoset Contribution Examples

The Element Information Item for the `a` element in the following document:

Example: Simple Link in http://www.example.org/test.xml

```
<doc>
<a xlink:href="foo.xml"
      xlink:show="new" xlink:actuate="onRequest">Click here</a>
</doc>
```

can be represented with one Link Information Item:

```
<link>
  <arc show="new" actuate="onRequest">
    <startParticipant resource="/doc/a[1]"/>
    <endParticipant resource="foo.xml"/>
  </arc>
</link>
```

For an information item that participates in several links, the situation is slightly more complex. Consider the following document:

Example: Extended Links in http://www.example.org/test2.xml

```
<doc>
<extendedlink xlink:type='extended'
      xlink:role='http://example.com/props/link1'>
  <loc xlink:type='locator' xlink:label='a' xlink:href='#a'></loc>
  <loc xlink:type='locator' xlink:label='b'
      xlink:role='http://example.com/props/loc1'
      xlink:href='#b1'></loc>
  <loc xlink:type='locator' xlink:label='b' xlink:href='#b2'></loc>
  <arc xlink:type='arc' xlink:from='a' xlink:to='b'
      xlink:arcrole='http://example.com/props/loc1'></arc>
</extendedlink>
<p><phrase id='a'>This</phrase> links to
<phrase id='b1'>this</phrase> and
<phrase id='b2'>this</phrase>.</p>
</doc>
```

In this document, there is one link with two arcs:

```
<link role="http://example.com/props/link1">
  <arc role="http://example.com/props/loc1">
    <participant resource="#a"/>
    <participant resource="#b1" role="http://example.com/props/loc1"/>
  </arc>
  <arc role="http://example.com/props/loc1">
    <participant resource="#a"/>
    <participant resource="#b2"/>
  </arc>
</link>
```

2.2 XLink Traversal

Link traversal always involves exactly two participating resources. In the context of a styling application, it can have several possible results:

1. The document currently being presented may be replaced by the new resource identified as the ending resource of the traversal.
2. A new viewing context (window, frame, pane, etc.) may be created in which to present the ending resource of the traversal.
3. The ending resource may be styled and the styled result may be embedded within the document currently being presented (instead of, or perhaps in addition to, the starting resource).
4. Application-specific semantics outside the scope of this note.

In the context of styling, each of these operations can be considered a presentation or style issue. This is particularly the case since XLink linking elements leave the processing of the attributes controlling presentation (`xlink:show` and `xlink:actuate`) at the discretion of the application.

Consequently, we will address these issues under style.

3 Styling Link Resources

Broadly speaking, style processing can be divided into two processes: styling of resources based upon how they participate in linking and styling the results of link traversal.

In this section, we consider the process of styling link resources. This model may be applied in two different ways:

1. Using current and anticipated technologies supported by existing W3C Recommendations (and Working Drafts, Candidate Recommendations, and Proposed Recommendations).
2. In an environment where the XSLT processor provides significantly more functionality for linking and contains several new features.

3.1 With Current Technologies

Once the XLink-aware application has identified the information items that are

participants in a link, styling the link becomes relatively straightforward. One can easily imagine an augmented CSS application or even a Perl script that made styling choices based upon the information obtained from the augmented Infoset.

Styling linking resources in XSL V1.0 can be achieved with small set of extension functions. The addition of a standard extension function API, anticipated for XSLT 1.1, makes this a practical solution for the present.

There are significant drawbacks to using extension functions for styling links, principal among them the fact that it places all of the burden on the stylesheet author. In order to handle links between arbitrary elements, *every* template would have to include conditional logic using the extension functions. This makes the extension function approach impractical as a long-term solution, but it is a solution that can be made to work today.

3.1.1 XSLT Extension Functions

In the descriptions that follow, `ext:` must be an extension function namespace as defined in [XSLT].

node-set **ext:link**()

The `ext:link` function returns a node set containing the Link Information Items known to the processor.

node-set **ext:arc-start**()

The `ext:arc-start` function returns a node set containing the Arc Information Items in the Starting Participant Arcs property of the context node. It takes no arguments.

node-set **ext:arc-end**()

The `ext:arc-end` function returns a node set containing the Arc Information Items in the Ending Participant Arcs property of the context node. It takes no arguments.

3.1.2 Function Example

The following templates show how one might format a `phrase` element in the source document differently depending on whether or not it was identified as the starting resource of a link:

```
<xsl:template match="phrase[count(ext:arc-start()) > 0]">
  <-- If the phrase is part of a link, make it a link to the target
         of the first arc. This does not adequately handle the case whe
         the phrase participates as the starting resource in more than
         one link. -->
  <fo:basic-link
      external-destination="{ext:arc-start()/endParticipant/@resource}"
    <xsl:apply-templates/>
  </fo:basic-link>
</xsl:template>

<xsl:template match="phrase">
```

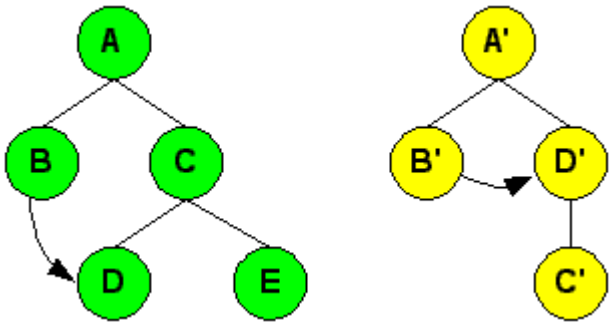
```
    <xsl:apply-templates/>
  </xsl:template>
```

3.2 With a Link-Aware XSLT Processor

[This section describes hypothetical additions to XSLT and/or XSL. These additions have not been reviewed by the WG and their presence here does not indicate that they will be added to any future version of XSL.]

In order to address the problems associated with placing all of the burden of link maintenance on the stylesheet author, we need to add XLink support into XSLT in a much more significant manner.

We observe that once the XSLT processor is aware of the link relationships that exist in the source document, it could conceivably maintain those relationships across transformation. Consider a simple document with a link from node B to node D:



During transformation, for every node in the result tree, the XSLT processor knows what the context node in the source tree was when the result node was created. It follows that XSLT can, in principle, maintain the link relationships across transformation.

Some difficulties need to be addressed:

1. How does the stylesheet author control the selection of links?

   Suppose that the source document links an element, A, in the source document with three other elements in the source document, B, C, and D. The XLink-aware processor can maintain that relationship in the result tree (linking A' with B', C', and D'), but what if the stylesheet author wishes to suppress the arc between A and C?

   We suggest two new top-level XSLT elements, `xsl:strip-links` and `xsl:preserve-links`, to provide this capability.
2. What happens if a source node is never processed, or produces no nodes in the result tree?

   If that source node is the sole participant at one end of an arc (or if all of the nodes in the participant resource have vanished), then that link does not occur in the result tree.
3. What happens if the source node is processed several times, producing disjoint nodes in the result tree?

We observe that heuristics have been described for this case in [HookerMaden2000]. Since heuristics are known to be imperfect, we suggest a new boolean attribute, `xsl:link-target`, described below.

4. How do you add new links to the Link Set model?

   Modelling the links as Infoset augmentation has the appealing property that it allows all of the links to be processed in the same way. But what if the stylesheet wants to add new links that should also be processed in the same way?

   We propose new XSLT elements for this purpose, see **3.3 Style Elements**.

The serializer for the result tree could instantiate the link relationships that it knows about using XLink, making this a nicely self-contained system: an XLink-aware XSLT processor produces a result tree that contains XLinks.

The problem of styling these links remains.

3.2.1 Controlling Link Selection

In some cases, stylesheet authors may wish to control the selection of links. We suggest two new top-level elements to do this: `xsl:strip-links` and `xsl:preserve-links`. As their names suggest, these work in a way roughly analagous to the way that `xsl:strip-space` and `xsl:preserve-space` work.

Each of these elements has a `select` attribute. The set of arcs selected by `xsl:strip-links` are not represented in the transformed document. The set selected by `xsl:preserve-links` are represented. Initially, all links are preserved.

If the select expression in either of these elements selects nodes other than Arc and Link elements in the Link Set, they are ignored.

3.2.2 Identifying Link Targets Explicitly

Presented with a complex transformation that produces several disjoint nodes (perhaps from different modes), it may be impossible for the XSLT processor to accurately identify the link targets automatically.

Although we expect heuristics to be used for most cases, we propose a new boolean attribute to make the selection explicit: `link-target`.

The `link-target` attribute can appear on `xsl:element`, and `xsl:text` elements and (as `xsl:link-target`) on literal result tree elements. When `true()`, the element on which it occurs is the link target for all links associated with the current context node. The use of `link-target` overrides any heuristic for the links which are associated with the context node.

Editorial note: Link Target Granularity	
Is this definition of `link-target` too coarse? It makes *all* links to the context node point to the nodes in the result tree identified as the link target. Might it be necessary to change this on a per-link basis?	

To better understand how this works, let’s consider a concrete example. Suppose that you have a link to a chapter title and that title appears in three different places in your result tree: in the Table of Contents, at the top of the page where the chapter begins, and in the running header on the pages of that chapter.

Imagine that the heuristics would select the inline at the top of the page where the chapter begins, but you wish to make a link to the entry in the table of contents. Conceptually, the heuristic algorithm must consider all of the possible link targets and select the "best one".

By adding a `xsl:link-target` to the literal result element that instantiates the chapter title in the Table of Contents, we give the the stylesheet author a way to inform the heuristic which element should explicitly be used, even if it is not the "best one" according to its algorithm.

3.2.3 Adding Style Properties

Automatic link identification is only half the battle. We still need to provide mechanisms for applying style to the transformed links.

There are two different types of styling that can be applied to links: property styling and structural styling. Property styling consists of those style changes that can be achieved by changing property values (or attributes) in the result tree. Structural styling consists of those style changes that require a restructured result tree.

Changes to fonts, foreground, and background colors are all things that can be achieved with property styling. Adding a horizontal rule above and below a resource, or putting a resource in a table, can only be achieved by structural styling.

We propose a new XSL element to provide style properties.

```
<!-- Category: instruction -->
<xsl:link-style
  select = expression
  xsl:use-attribute-sets = qnames>
  <!-- Content: xsl:attribute* -->
</xsl:link-style>
```

The select expression of `xsl:link-style` should always select Link Information Items. The attributes provided are associated with the links selected.

3.2.3.1 Style Properties Example

The following templates show how one might make starting resources with the "HTML" role appear blue and underlined in the result tree.

```
<xsl:link-style select="link::*/@role='http://www.w3.org/1999/xhtml'">
  <xsl:attribute name="color">blue</xsl:attribute>
  <xsl:attribute name="text-decoration">underlined</xsl:attribute>
```

3.2.4 Style-based Structural Changes

In order to support structural styling, it is essentially necessary for a stylesheet to apply more than one template to a given source node. At the moment, this can be accomplised with `xsl:import` and `xsl:apply-imports`. It has already been suggested that a general facility be provided for this feature.

For example, `xsl:next-match` could be added to XSLT. It would apply the next-highest-priority template to the current context node. Using next match, we could put links in HTML tables as follows:

```
<xsl:template match="*[count(arc-start::*) > 0]" priority="100">
  <table>
    <tr>
      <td>
        <xsl:next-match/>
      </td>
    </tr>
  </table>
</xsl:template>
```

By assigning this template a very high priority, we can assure that it will always match first. The `xsl:next-match` inside will apply the next highest priority template so that the formatting for each individual element is correct.

Consider our earlier example of a styling a phrase that participated in a link. While we previously had to add conditional logic to all of our phrase-level elements, using `xsl:next-match`, we can handle all of them simultaneously.

3.2.4.1 Next Match Example

The following templates show how one might format elements in the source document differently depending on whether or not they are identified as the starting resource of a link:

```
<xsl:template match="*[arc-start::*]" priority="100">
  <-- If the element is part of a link, make it a link to the target
         of the first arc. This does not adequately handle the case whe
         the phrase participates as the starting resource in more than
         one link. -->
  <fo:basic-link
      external-destination="{arc-start::*[1]/endParticipant/@resource}"
    <xsl:next-match/>
  </fo:basic-link>
</xsl:template>

<xsl:template match="phrase">
  <xsl:apply-templates/>
```

```
  </xsl:template>

  <xsl:template match="emphasis">
    <em><xsl:apply-templates/></em>
  </xsl:template>

  <!-- etc. -->
```

3.3 Style Elements

Our original goal in writing this note was to explore how linking and style effect each other and how a processing system might reasonably deal with these issues. Several early reviewers suggested that this work could be taken further to provide a mechanism not only for providing uniform treatment of source document links, but also uniform treatment of links created by the stylesheet. In this section, we explore some of those ideas.

Consider this simple document:

```
  <doc>
  <p><name>Pat Jones</name> is taking <course xlink:type='simple'
        xlink:href='courses/CS101.xml'>CS-101</course>.</p>
  </doc>
```

Imagine for the moment that our goal is to link the persons name with some online address book. One way to do that would be to construct the link natively in the result tree vocabulary with a template like this (assuming our target is HTML):

```
  <xsl:template match="name">
    <a href="http://example.com/address?{.}">
      <xsl:apply-templates/>
    </a>
  </xsl:template>
```

But that has the unsatisfactory result that the link is unknown to the "link aware" processor we've been describing. That processor has already constructed a link set for the source document:

```
  <linkset xmlns="&namespaceName;">
    <link>
      <arc>
        <startParticipant resource="/doc/p/course[1]"/>
        <endParticipant resource="courses/CS101.xml"/>
      </arc>
    </link>
  </linkset>
```

What would seem more appropriate would be to insert the resulting link into the "result document link set". The result document link set is conceptually analagous to the source document link set: it describes the linking relationships in the result document.

Up to now, the result document link set has been discussed only implicitly because it was derived solely from the source document link set.

We introduce four new XSL elements for constructing links in the result tree link set: `xsl:link`, `xsl:arc`, `xsl:start-participant`, and `xsl:end-participant`.

3.3.1 xsl:link

```
<xsl:link
  id = id>
  <!-- Content: (xsl:arc+, other-namespace-elements) -->
</xsl:link>
```

An `xsl:link` element selects or constructs a link in the result tree link set. If an ID is given, it is logically an ID in the result tree link set. By using the same ID value, it is possible for different templates to add arcs to the same `xsl:link`.

Additional elements in other namespaces can be provided, the effect that they have on rendering the link is undefined.

3.3.2 xsl:arc

```
<xsl:arc
  role = uri-reference
  title = string
  show = "new" | "replace" | "embed" | "other" | "none"
  actuate = "onLoad" | "onRequest" | "other" | "none">
  <!-- Content: (xsl:start-participant, xsl:end-participant, other-
namespace-elements*) -->
</xsl:arc>
```

An `xsl:arc` adds an arc to the result tree link set.

Additional elements in other namespaces can be provided, the effect that they have on rendering the link is undefined.

3.3.3 xsl:start- and xsl:end-participant

```
<xsl:start-participant
  select = node-set-expression
  resource = uri-reference>
  <!-- Content: (other-namespace-elements*) -->
</xsl:start-participant>
```

```
<xsl:end-participant
  select = node-set-expression
  resource = uri-reference>
  <!-- Content: (other-namespace-elements*) -->
</xsl:end-participant>
```

The `xsl:start-participant` and `xsl:end-participant` elements identify the beginning and ending resources of the arc. Each resource can be identified by *either* a `select` or a `resource` attribute.

3.3.4 Using the Link Elements

With these elements, we can recast our earlier template as follows:

```
<xsl:template match="name">
  <xsl:link>
    <xsl:arc>
      <xsl:start-participant select="."/>
      <xsl:end-participant resource="http://example.com/address?{.}"/>
    </xsl:arc>
  </xsl:link>
</xsl:template>
```

3.3.5 Merging Link Sets

The links created by `xsl:link` are placed in the result tree link set. They are explicitly not placed in the source tree link set and are not available for querying by the stylesheet.

If an explicit result tree link set is desired, when the styling process completes, it must be constructed by merging the relevant links from the source document link set (with appropriate translation into the result tree) with the links explicitly created in the result document link set.

If this is our result tree:

```
<div>
  <p><em>Pat Jones</em> is taking <span>CS 101<span></p>
</div>
```

Then the source document link set would contribute the following links to the result document link set:

```
<linkset xmlns="&namespaceName;">
  <link>
    <arc>
      <start-participant resource="result-tree:/div/p/span[1]"/>
      <end-participant resource="courses/cs101.xml"/>
    </arc>
  <link>
```

Note:

We've introduced the "scheme" `result-tree:` to express pointers into the result tree. This is just a convention for presentation here. In a real implementation, pointers or some other mechanism would more likely be used and if the link set were serialized, a proper base URI would be known.

The template that constructed an explicit link would contribute:

```
<linkset xmlns="&namespaceName;">
  <link>
    <arc>
      <start-participant resource="result-tree:/div/p/em[1]"/>
      <end-participant resource="http://example.com/address?Pam Jones"/>
    </arc>
  <link>
```

So the resulting link set is `linkset` that contains these two `link`s.

3.3.6 What's Missing

These elements allow the stylesheet author to construct links that become part of the unified processing model. One obvious weakness that remains is that it is not possible for these elements to link to generated text. Suppose that I have a cross reference element that is processed as follows:

```
<xsl:template match="xref">
  <xsl:text>See also: </xsl:text>
  <xsl:apply-templates/>
</xsl:template>
```

There's no way to use the `xsl:link` elements to construct a link from the generated text "See also: ", because there's no source tree select expression that can identify it.

One possible way to move forward would be to add a new attribute, `result-tree-id` which would allow the generated text to be addressed:

```
<xsl:template match="xref">
  <xsl:text result-tree-id="xref{generate-id(.)}">See also: </xsl:text>
  <xsl:apply-templates/>
  <xsl:link>
    <xsl:arc>
      <xsl:start-participant select="result-tree:/id(xref{generate-id(.)
      <xsl:end-participant resource="whatever-is-appropriate"/>
    </xsl:arc>
  </xsl:link>
</xsl:template>
```

But this seems contrived and unsatisfactory. We invite suggestions.

4 Styling Link Traversal

After much consideration, we arrived at the conceptual model which follows. This model exposes a significant issue in the interaction between the current technologies for linking and styling: [XPointer] can address documents at the character level whereas [XSLT] can only address documents at the text-node level.

We show that by selecting specific defaults within the model, we can provide

expected functionality with current technology. As the technology of linking and styling matures, we'll be able to exploit more of the model with standard technologies. (Implementations will be able to exploit the entire model immediately by employing extensions to standard functionality.)

4.1 A Conceptual Model for Styling Link Traversal

From a styling perspective, link traversal requires the styling application to retrieve the document which contains the ending resource, style it, and present it. In addition, the ending resource may get special treatment. It might, for example, be shown in reverse video or the document view might be adjusted so that the ending resource appears at the top of the viewing context.

Displaying the styled result is a complex problem because links are between *source documents*, not between their styled results. Consider, for example, an XLink cross-reference:

```
See <xref xlink:type="simple" xlink:href="#ch5">Chapter 5</xref>
```

This link expresses a relationship between the text "Chapter 5" and the element with an ID of "ch5" elsewhere in the document. It provides no specific information about the formatted result that might be presented to the reader. Online, this might become a hypertext link to some other location whereas in print it might be rendered simply as a page number in parenthesis (or not at all).

The arbitrary nature of styling transformations can make this problem even more complex, as when the target in the source document is rendered in multiple locations in the presented result (See **5 Discontiguous Resources**).

We may also want to consider how much of the document that contains the ending resource we wish to display. HTML browsers traditionally retrieve the entire resource and display it, moving the viewing context to the point where the ending resource occurs. In the case of XML documents, we may wish to have more control. We add one additional generalization to the conceptual model for this purpose.

Rather than presenting the entire resource, we might want to display a subresource within the document *that is different* from the subresource that is the ending resource of the link. For example, to style a link to a figure within a book, we might want to display the whole chapter that contains the figure:

```
<doc id='process'>
  <chap>...</chap>
  <chap>...</chap>
  <chap id='present'>
    <p>...</p>
    <figure id='target'>
      ...
    </figure>
    <p>...</p>
    <p>...</p>
  </chap>
  <chap>...</chap>
```

```
</doc>
```

In other words, we might wish to style a link by processing a whole document (in order to get chapter numbering correct, for example), present the third chapter in the user agent, and highlight the figure.

We describe this in terms of three distinct contexts, each identified by a URI reference (probably including an [XPointer] fragment identifier in most cases):

1. [Definition: The **Processing Context** of a link traversal is the resource (or portion of a resource) that is available to the style engine.]

   Informally, this is the tree that is available to the styling application. Usually this is the whole document.
2. [Definition: The **Presentation Context** of a link traversal is the resource (or portion of a resource) that is effectively the source resource to be styled.]

   In other words, the results of styling the [Presentation Context] is the content that is presented in the user agent. Often this is the whole document.
3. [Definition: The **Target** of a link traversal is the ending resource of the link.]

These components, plus the stylesheet to be used, are present in the Infoset contributions described in **2.1.1 XLink Infoset Contributions**. However, the current XLink specification does not provide standard ways to provide all of these components. Rather than request possibly contentious additions to XLink at this late date, we recommend using default values for several components.

For embedded resources, we recommend that the default processing and presentation contexts be the same as the target (see **6.2 Ranges that are Presentation Context**). For resources displayed with "new" or "replace" semantics, we recommend that the default processing and presentation contexts be the whole document that contains the ending resource.

In any case, the application has considerable freedom in selecting the appropriate stylesheet. In some applications it may make sense to use the same stylesheet as the source document (particularly, perhaps, in the "embed" case); in others it may make sense to use the XML Stylesheet PI ([XML Stylesheet PI]) from the target document or some other application-specific selection.

Application-specific behavior can be modeled with extension metadata in the XLink linking elements. If useful subsets of this behavior become common, a future version of XLink may promote them to standard elements.

4.1.1 Special Considerations for Embedded Resources

The model articulated here is the most consistent and rational model that we could imagine. It does not, however, produce exactly the results that might be expected in all cases. In particular, embedding resources requires special consideration. (Embedding resources identified by XPointer ranges is even worse; see **6.2 Ranges that are Presentation Context**.)

Consider the following documents, `doc1.xml`:

```
<doc>
<?xml-stylesheet href="style1.xsl" type="text/xsl"?>
<list>
<item>x xxxx</item>
<item>xxx xx xxxx</item>
<embed xlink:type="simple"
       xlink:show="embed" xlink:actuate="onLoad"
       xlink:href="doc2.xml#xpointer(//list[1]/item[2])"/>
<item>xxxxx</item>
</list>
</doc>
```

and `doc2.xml`:

```
<doc>
<?xml-stylesheet href="style2.xsl" type="text/xsl"?>
<list>
<item>y yyyy</item>
<item>yyy yy yyy</item>
<item>yyyy</item>
</list>
</doc>
```

Suppose that `style1.xsl` styles lists with Arabic numerals and `style2.xsl` styles them with Roman numerals (and no other significant transformations are performed). What will the results be?

If the embedded fragment is processed with `style1.xsl`, the result will be something like this:

```
1. x xxxx
2. xxx xx xxxx
2. yyy yy yyy
4. xxxxx
```

(In fact, it is even possible that the last item will be numbered "3" if the stylesheet is only counting `<item>`s, but that can be avoided in the stylesheet.)

If the embedded fragment is processed with `style2.xsl`, the result will be something like this:

```
1. x xxxx
2. xxx xx xxxx
ii. yyy yy yyy
4. xxxxx
```

In neither case will the result that the author probably had in mind (namely 1, 2, 3, 4) be the result.

In fact, the desired result can be achieved, but only by constructing the *source* document in an appropriate way:

```
<doc>
<?xml-stylesheet href="style1.xsl" type="text/xsl"?>
<list>
<item>x xxxx</item>
<item>xxx xx xxxx</item>
<item><embed xlink:type="simple"
       xlink:show="embed" xlink:actuate="onLoad"
       xlink:href="doc2.xml#xpointer(//list[1]/item[2]/node())"/>
</item>
<item>xxxxx</item>
</list>
</doc>
```

By selecting only the content of the target list item, we allow the stylesheet for the source document to number all of the items in a logical way.

This is hardly the best possible situation, but no superior alternative seems evident.

Since both of these documents seem to use the same tag set, it may have been closer to the author's original intention to style a document that contained *a merged source document* (as might be achieved with [XInclude]) rather than *merging the styled results* (as is achieved with [XLink]). See **4.1.3 XInclude vs. XLink Embedding**.

4.1.2 Result Tree Numbering

To compound the situation further, the two most popular presentation formats for XML documents, HTML and XSL Formatting Objects, perform numbering in completely different ways.

In the HTML, list item numbering is actually done by the formatter, whereas it is done by the transformation that produces FOs in XSL. This means that the first example described above will actually work for HTML but not for XSL FOs.

Conversely, if the desired result was a mixture of numbering systems (as might be the case in a legal document, perhaps), the former result could not be achieved in HTML without considerable effort.

Consider how the stylesheets would likely produce formatting objects for the `doc1.xml`:

```
<list-block>
<list-item>
  <list-item-label><block>1.</block></list-item-label>
  <list-item-body><block>x xxxx</block></list-item-body>
</list-item>
<list-item>
  <list-item-label><block>2.</block></list-item-label>
  <list-item-body><block>xxx xx xxxx</block></list-item-body>
```

```
    </list-item>
    <list-item>
      <list-item-label><block>2.</block></list-item-label>
      <list-item-body><block>yyy yy yyy</block></list-item-body>
    </list-item>
    <list-item>
      <list-item-label><block>4.</block></list-item-label>
      <list-item-body><block>xxxxx</block></list-item-body>
    </list-item>
    </list-block>
```

As you can see, the list item numbers are textually part of the result tree. This is not so in the HTML case:

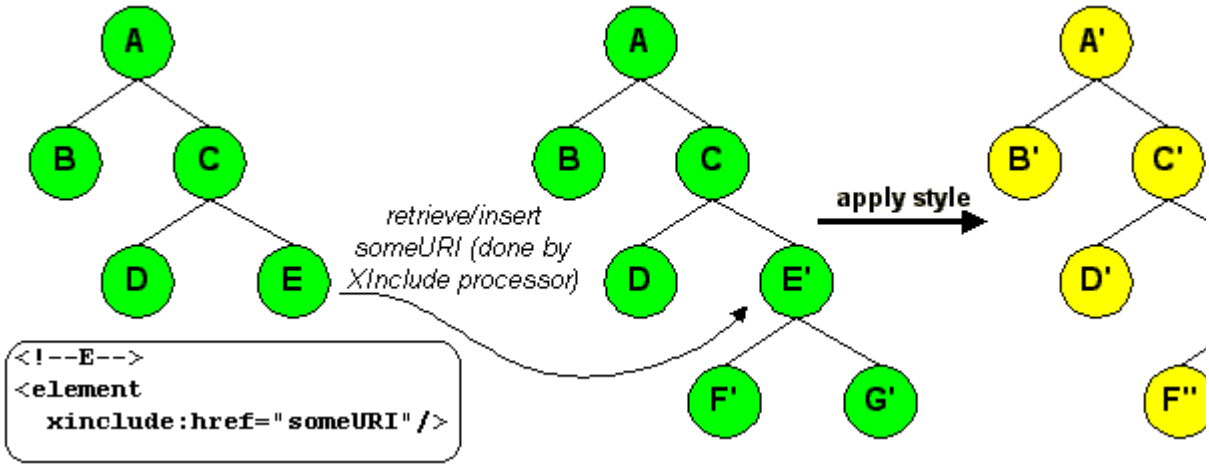
```
    <ol>
    <li>x xxxx</li>
    <li>xxx xx xxxx</li>
    <li>yyy yy yyy</li>
    <li>xxxxx</li>
    </ol>
```

The list items are enumerated by the browser in the HTML case, so the numbering will be sequential.

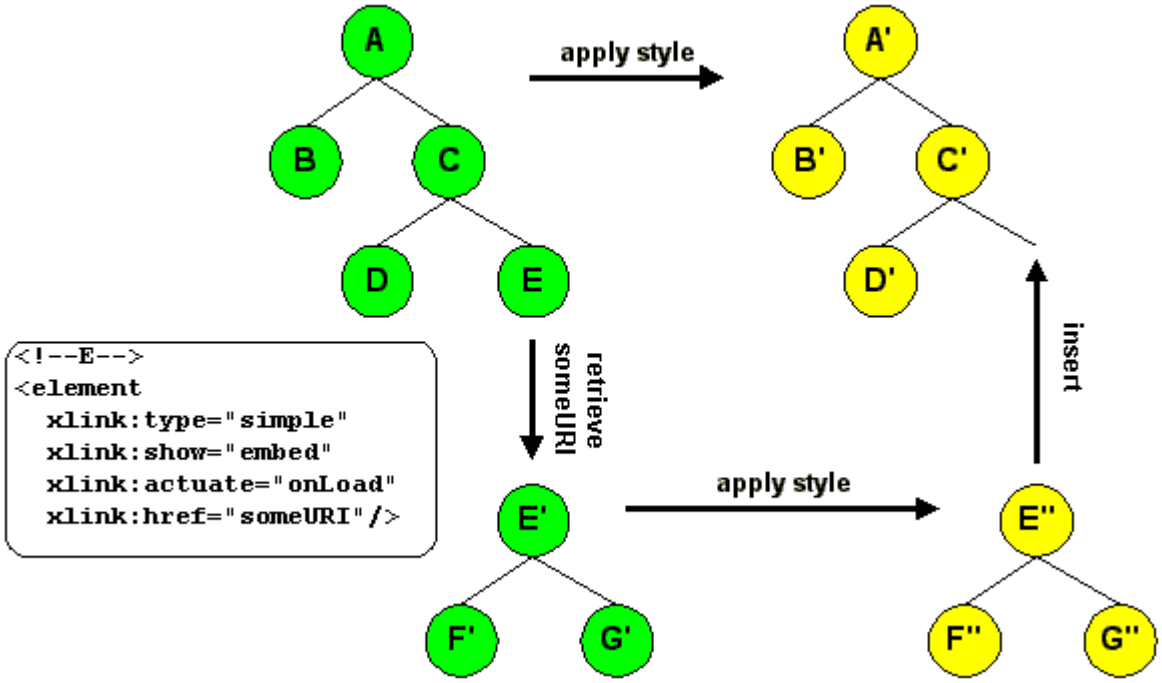
4.1.3 XInclude vs. XLink Embedding

The distinction between [XInclude] and [XLink] deserves special consideration.

At a high level, [XInclude] is similar to external entity references in [XML]. That is, the inclusion is done at or shortly after parsing, before style is applied. As you can see in the following figure, the element "E" is identified as an XInclude element and the content *of the source tree* is modified *prior* to styling:



Embedded XLinks, as we've described them, are handled during the styling process. When an embedded XLink is processed, the *result of styling the ending resource* of the link *is merged* into the *result of styling the document into which it is embedded*.



The decision about whether to use an [XInclude] or an embedded [XLink] is very dependent on the application. As a general rule, if the resource being embedded is from the same schema as the embedding document, and it will be styled with the same stylesheet, [XInclude] is probably a good choice. If the embedded document is from a different schema, or requires a different stylesheet, [XLink] is probably the right choice.

4.2 The Semantics of Nested Linking

The actuate axis of an XLink linking element has two standard, defined values, `onRequest` and `onLoad`. For links which have the `onRequest` semantics, style processing is straightforward: processing continues (possibly to completion) until a user event initiates the link.

For links which have the `onLoad` semantic, however, things are slightly more complicated. The styling application is expected to process these links during the styling process (effectively, "when the document is loaded").

There are several cases to consider:

An "onLoad" link with an `xlink:show` of "new"

> The styling application should create a new viewing context (window, frame, pane, etc.) and display the ending resource of the link in that context.

An "onLoad" link with an `xlink:show` of "replace"

> The styling application should replace the document currently being styled with the ending resource of the link.
>
> The [XLink] specification says that the first such link in document order is the

one that should be processed. However, because a styling application has the freedom to process the nodes of a document in any order, this definition seems restrictive. We recommend instead that it simply state that the first such link encountered should be processed. (Subsequent nodes will clearly not be processed since the original document is abandoned.) By extension, if the first link encountered has multiple arcs, the first arc should be processed.

Note:

The fact that this behavior can lead to infinite loops through one or more documents is a problem outside the scope of this note.

An "onLoad" link with an `xlink:show` of "embed"

The styling application should embed the result of styling the ending resource of the link in the current document.

The possibility of a resource that is *being embedded* containing an "onLoad" link must also be considered:

An "onLoad" link with an `xlink:show` of "new" encountered in a resource being embedded

The styling application should create a new viewing context (window, frame, pane, etc.) and display the ending resource of the link in that context.

In other words, if A embeds B which contains an onLoad/new link to C, the result will be B embedded in A in one viewing context and C in another viewing context.

An "onLoad" link with an `xlink:show` of "replace" encountered in a resource being embedded

The style processor should replace the *resource being embedded* with the result of styling the ending resource of the link.

In other words, if A embeds B which contains an onLoad/replace link to C, the result will be C embedded in A, with no trace of B.

An "onLoad" link with an `xlink:show` of "embed" encountered in a resource being embedded

The styling application should embed the result of styling the ending resource of the link in the current document.

In other words, if A embeds B which contains an onLoad/embed link to C, the result will be C embedded in B embedded in A.

5 Discontiguous Resources

Styling discontiguous resources presents a special set of problems. One set of problems arises because we may wish to offer behavior in the presentation engine that is independent of whether or not the resource is discontiguous. A more significant issue involves the way in which style transformations may create discontiguous resources from a single resource in the source document.

5.1 Implementing Consistent Behavior

Some presentation systems implement special behaviors with respect to linking resources. For example, browsers may present links in a different color or speak them in a different tone of voice.

Often, discontiguous resources can be viewed as distinct. That is, a single resource consisting of three phrases can generally be treated like three resources, each consisting of a single phrase.

When this is not sufficient, for example, when the stylesheet author wishes to implement "mouse-over" highlighting of links and she wants discontiguous parts of the same link to be highlighted together, additional effort may be required.

The enhanced Infoset constructed by XLink identification, provides the raw materials from which such a feature could be implemented.

The XSL Formatting Objects include properties for selecting alternate presentations based on conditions. These properties, in conjunction with a presentation system that has access to the enhanced Infoset would allow an application to support this behavior.

Likewise for other presentation systems, access to the enhanced Infoset provides enough information to implement the required functionality.

5.2 Discontinuity Introduced by Styling

Transforming source documents for presentation may introduce discontinuities. For example, consider a link to a chapter title. In the document that contains the ending resource, that title is a single, unambiguous element. But when that document is styled, the chapter title may occur in several places (in the table of contents, on the chapter title page, and in a dozen cross references). Which one of these represents the ending resource?

No perfect algorithm for finding the target is readily apparent, but reasonable heuristics have been described [HookerMaden2000].

To the extent that it is practical, we think applications should provide users with access to all of the possible targets.

6 Ranges are Challenging

XPointer ranges present a number of exceptional challenges to the styling process. In the simplest case, they may be used to style components that are

smaller than the components exposed by the data model of a particular styling language. Consider this link:

```
<doc>
<p>Some text about Pat Jones that participates in a third
party link.</p>
</doc>
```

The link is composed only of characters in the middle of a text node.

Ranges can also span normal element boundaries, producing ranges of text that are not well-formed in the XML sense:

```
<doc>
<p>This text demonstrates
<emph>a link</emph> that spans a <emph>not well-formed</emph>
range.</p>
</doc>
```

6.1 Styling Ranges in a Larger Presentation Context

Styling ranges (starting or ending resources expressed with XPointer ranges) in a larger context is problematic when the data model of the style language has a coarser granularity than the XML character. In particular, the XSLT 1.0 data model exposes text nodes, but not individual characters.

We propose that a reasonable processing model in this case is simply to style the smallest available node in the data model. For example, we expect `ext:arc-start()` to return a non-empty node set on every text node that contains one or more characters that are starting resources.

This will generally result in the "sloppy" application of style. For example,

```
<doc>
<p>Some text about Pat Jones that participates in a third
party link.</p>
</doc>
```

will be styled as if the link were:

```
<doc>
<p>Some text about Pat Jones that participates in a third
party link.</p>
</doc>
```

and

```
<doc>
<p>This text demonstrates
<emph>a link</emph> that spans a <emph>not well</emph>-formed</emph>
range.</p>
</doc>
```

will be styled as if the link were:

```
<doc>
<p>This text demonstrates
<emph>a link</emph> that spans a <emph>not well-formed</emph>
range.</p>
</doc>
```

If styling markup of this sort is a requirement for XSLT 2.0, as we think it is, then the data model of XSLT will have to be extended to include characters.

A few additional extension functions could be added which would offer a laborious solution to a subset of the range problems that can occur inside a text node through extensive use of the `substring` function. But it is not clear that any rational substring-based approach will handle all of the possible cases (e.g. discontiguous character ranges within a single text node that taken together form a linking resource).

6.2 Ranges that are Presentation Context

The problem of ranges in the presentation context is much more serious. When the presentation context is a range, a "sloppy" approach is unsuitable. If the link author requests that a specific range of the target document is to be presented, displaying more or or less than that range may be disastrous. Consider the case where a warning is accidentally lost or an adjacent range that has security implications is accidentally displayed.

The problem is especially troublesome when we consider embedded links. In the embedded case, we have suggested that the presentation context be the same as the target. If an embedded link uses ranges to point to a portion of a text node, it would be entirely unacceptable to embed the entire text node.

> **Note:**
>
> Because the "sloppy" approach is unsuitable in this case, it follows that ranges cannot be implemented using tools that have a data model that does not address individual characters.

To tackle the difficult case introduced by ranges, we suggest that a subset of XPointer that did not include ranges would be a useful product. Such a subset could then be used as the data type for presentation context and `show="embed"` ending resources.

However, we suggest the following approach of "pruning" as a possible work-

around for applications which must support presentation contexts that are defined in terms of ranges:

1. Identify the smallest contiguous set of whole information items which completely contains the resource in question:

```
<doc>
<p>This text demonstrates
<emph>a link</emph> that spans a <emph>not well</emph>-formed</emph>
range.</p>
</doc>
```

2. Within this set, it is possible to identify three classes of information items:
    1. Those that are wholly within the range: e.g. all the character information items that occurs between the two `emph` elements.
    2. Those that are wholly outside the range: e.g. the character information items that comprise the string "-formed".
    3. Those that are partially within the range: e.g. the `emph` Element Information Items.
3. Remove all information items that are wholly *outside* the range:

```
<emph>link</emph> that spans a <emph>not well</emph>
```

    In this example, only text was removed, but if additional elements had been included by the initial set of information items, they would have been removed as well.

The node set that remains after this process becomes *both* the presentation context *and* the processing context for the link.

It is necessary for the processing context to be restricted as well as the presentation context because there is no practical way, in the general case, under arbitrary transformation, for an application to trim the *styled* result to just the nodes that resulted from processing the target. The tree must be pruned *before* styling. We feel that this additional complication provides further evidence in favor of a range-free subset of XPointer.

Note:

Pruning prior to styling introduces the possibility that nodes which contributed to the style of the result may have been lost and therefore unexpected styling may occur.

6.3 Ranges and Glyph/Character Boundaries

Ranges can also be problematic if they have boundaries that occur at certain glyph and character locations.

Consider the following range:

```
<doc>
<p>...a brief history of flying machines.</p>
</doc>
```

In a visual presentation system, what is the extent of that link if the word "flying" begins with the "fl" ligature?

This problem is not limited to subtle issues of appropriate glyph selection for aesthetic purposes. In some Indic languages, for example, the actual order of the displayed glyphs may be different from the order of characters in the source. Similar problems occur in Thai, Arabic, and Hebrew, where the order and position of displayed glyphs depends on linguistic factors, not simply on the order of the characters in the source document.

It's unclear if there are any specific recommendations that can be made about this issue globally, except that avoiding character ranges, especially ranges that occur inside words, is probably best.

A References

The distinction between normative and non-normative references is made only to emphasize the distinction between references to stable documents (W3C Recommendations, IETF RFCs, etc.) and documents that are in some sense unstable (W3C Working Drafts, etc.). W3C Notes have no normative status whatsoever.

A.1 Normative References

XML
Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen, editors. *Extensible Markup Language (XML) 1.0*. World Wide Web Consortium, 1998. (See http://www.w3.org/TR/REC-xml.)

XML Names
Tim Bray, Dave Hollander, and Andrew Layman, editors. *Namespaces in XML*. World Wide Web Consortium, 1999. (See http://www.w3.org/TR/REC-xml-names/.)

XPath
James Clark, Steve DeRose, editors. *XML Path Language (XPath) Version 1.0*. World Wide Web Consortium, 2000. (See http://www.w3.org/TR/xpath.)

XSLT
James Clark, editor. *XSL Transformations (XSLT) Version 1.0*. World Wide Web Consortium, 1999. (See http://www.w3.org/TR/xslt.)

XML Stylesheet PI
James Clark, editor. *Associating Style Sheets with XML Documents Version 1.0* World Wide Web Consortium, 1999. (See http://www.w3.org/TR/xml-stylesheet/.)

IETF RFC 2396
IETF (Internet Engineering Task Force). *RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax*. T. Berners-Lee, R. Fielding, L. Masinter. 1998. (See http://www.ics.uci.edu/pub/ietf/uri/rfc2396.txt.)

A.2 Other References

HookerMaden2000
Hooker, Deborah, and Christopher Maden. *Æsop: A Browser for XML Documents and Open eBook Publication Structures*. In XTech 2000, San José, February 2000. (See http://www.oreilly.com/%7Ecrism/xtech2k/.)

XSL
Sharon Adler, Anders Berglund, Jeff Caruso, et. al., editors. *Extensible Stylesheet Language (XSL) Version 1.0*. World Wide Web Consortium, 2000. (See http://www.w3.org/TR/xsl/.)

XML Base
Jonathan Marsh, editor. *XML Base*. World Wide Web Consortium, 2000. (See http://www.w3.org/TR/xmlbase/.)

XInclude
Jonathan Marsh, David Orchard, editors. *XML Inclusions (XInclude) Version 1.0*. World Wide Web Consortium, 2000. (See http://www.w3.org/TR/xinclude/.)

XML Infoset
John Cowan, editor. *XML Information Set*. World Wide Web Consortium, 2000. (See http://www.w3.org/TR/xml-infoset/.)

XLink
Steve DeRose, Eve Maler, David Orchard, Ben Trafford, editors. *XML Linking Language (XLink) Version 1.0*. World Wide Web Consortium, 2000. (See http://www.w3.org/TR/xlink/.)

XPointer
Ron Daniel Jr., Steve DeRose, Eve Maler, editors. *XML Pointer Language (XPointer) Version 1.0*. World Wide Web Consortium, 2000. (See http://www.w3.org/TR/xpath.)

XML Schema (Structures)
Henry S. Thompson, David Beech, Murray Maloney, et. al. editors. *XML Schema Part 1: Structures*. World Wide Web Consortium, 2000. (See http://www.w3.org/TR/xmlschema-1/.)

B Summary of Proposed Changes to XSLT (Non-Normative)

This appendix summarizes the proposed changes to XSLT/XSL to provide better support for linking.

B.1 link-target

The `link-target` attribute can appear on `xsl:element` elements and on literal result tree elements. When `true()`, the element on which it occurs is the link target for any links associated with the current context node. The use of `link-target` overrides any heuristic for the links which are associated with the context node.

B.2 xsl:link-style

We propose a new XSL element to provide style properties.

```
<!-- Category: instruction -->
<xsl:link-style
  select = expression
  xsl:use-attribute-sets = qnames>
  <!-- Content: xsl:attribute* -->
</xsl:link-style>
```

The select expression of `xsl:link-style` should always select Link Information Items. The attributes provided are associated with the links selected.

B.3 XSL Linking Elements

The `xsl:link`, `xsl:arc`, `xsl:start-participant`, and `xsl:end-participant` elements allow a stylesheet author to construct links that can be treated uniformly with links that are manifest in the source or in a link base.

B.4 xsl:next-match

The `xsl:next-match` element selects the next highest priority template for the current context node and applies it.

C W3C XLink/XSL Joint Task Force (Non-Normative)

This note was prepared and approved for publication by the W3C XLink/XSL Joint Task Force (TF). TF approval of this note does not necessarily imply that all TF members agreed with all aspects of the note. The members of the XLink/XSL Joint TF are:

- Sharon Adler , IBM
- Anders Berglund , IBM
- Paul Grosso , Arbortext
- Eduardo Gutentag , Sun
- Chris Maden
- Eve Maler , Sun (*Chair*)
- Norman Walsh , Sun (*Editor*)