XML Inclusions (XInclude)

Version 1.0 (Second Edition)

W3C Recommendation 15 November 2006

This version:

http://www.w3.org/TR/2006/REC-xinclude-20061115/

Latest version:

http://www.w3.org/TR/xinclude/

Previous versions:

http://www.w3.org/TR/2006/PER-xinclude-20061003/

Authors and Contributors:

Jonathan Marsh (Microsoft) <jmarsh@microsoft.com>
David Orchard (BEA Systems) <dorchard@bea.com>
Daniel Veillard <daniel@veillard.com>

Abstract

This document specifies a processing model and syntax for general purpose inclusion. Inclusion is accomplished by merging a number of XML information sets into a single composite infoset. Specification of the XML documents (infosets) to be merged and control over the merging process is expressed in XML-friendly syntax (elements, attributes, URI references).

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This document has been produced by the W3C XML Core Working Group as part of the XML Activity. The English version of this specification is the only normative version. However, for translations of this document, see http://www.w3.org/2003/03/Translations/byTechnology?technology=xinclude10.

This document is a W3C Recommendation. This second edition is not a new version of XInclude. As a convenience to readers, it incorporates the changes dictated by the accumulated errata (available at http://www.w3.org/2004/12/xinclude-errata/ to the First Edition of XInclude 1.0, dated 20 December 2004, which it supersedes.

Please report errors in this document to the public www-xml-xinclude-comments@w3.org mailing-list; archives are available. The errata list for this edition is available at http://www.w3.org/XML/2006/11/xinclude-errata/.

Known implementations are documented in the XInclude Implementation Report at http://www.w3.org/XML/2004/xinclude-implementation/report.html. A Test Suite is maintained at http://www.w3.org/XML/Test/XInclude/ to help in assessing conformance to this specification. The latest release of the Test Suite includes new test cases which implementers can use to check their conformance to the changes included in this new edition.

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document is governed by the 24 January 2002 CPP as amended by the W3C Patent Policy Transition Procedure. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

Appendices

1. Introduction

Many programming languages provide an inclusion mechanism to facilitate modularity. Markup languages also often have need of such a mechanism. This specification introduces a generic mechanism for merging XML documents (as represented by their information sets) for use by applications that need such a facility. The syntax leverages existing XML constructs - elements, attributes, and URI references.

1.1. Relationship to XLink

XInclude differs from the linking features described in the [XML Linking Language], specifically links with the attribute value `show="embed"`. Such links provide a media-type independent syntax for indicating that a resource is to be embedded graphically within the display of the document. XLink does not specify a specific processing model, but simply facilitates the detection of links and recognition of associated metadata by a higher level application.

XInclude, on the other hand, specifies a media-type specific (XML into XML) transformation. It defines a specific processing model for merging information sets. XInclude processing occurs at a low level, often by a generic XInclude processor which makes the resulting information set available to higher level applications.

Simple information item inclusion as described in this specification differs from transclusion, which preserves contextual information such as style.

1.2. Relationship to XML External Entities

There are a number of differences between XInclude and [XML 1.0] or [XML 1.1] external entities which make them complementary technologies.

Processing of external entities (as with the rest of DTDs) occurs at parse time. XInclude operates on information sets and thus is orthogonal to parsing.

Declaration of external entities requires a DTD or internal subset. This places a set of dependencies on inclusion, for instance, the syntax for the DOCTYPE declaration requires that the document element be named - orthogonal to inclusion in many cases. Validating parsers must have a complete content model defined. XInclude is orthogonal to validation and the name of the document element.

External entities provide a level of indirection - the external entity must be declared and named, and separately invoked. XInclude uses direct references. Applications which generate XML output incrementally can benefit from not having to pre-declare inclusions.

Failure to load an external entity is normally a fatal error. XInclude allows the author to provide default content that will be used if the remote resource cannot be loaded.

The syntax for an internal subset is cumbersome to many authors of simple well-formed XML documents. XInclude syntax is based on familiar XML constructs.

1.3. Relationship to DTDs

XInclude defines no relationship to DTD validation. XInclude describes an infoset-to-infoset transformation and not a change in XML parsing behavior. XInclude does not define a mechanism for DTD validation of the resulting infoset.

1.4. Relationship to XML Schemas

XInclude defines no relationship to the augmented infosets produced by applying an XML schema. Such an augmented infoset can be supplied as the input infoset, or such augmentation might be applied to the infoset resulting from the inclusion.

1.5. Relationship to Grammar-Specific Inclusions

Special-purpose inclusion mechanisms have been introduced into specific XML grammars. XInclude provides a generic mechanism for recognizing and processing inclusions, and as such can offer a simpler overall authoring experience, greater performance, and less code redundancy.

2. Terminology

The key words *must*, *must not*, *required*, *shall*, *shall not*, *should*, *should not*, *recommended*, *may*, and *optional* in this specification are to be interpreted as described in [IETF RFC 2119].

The term *information set* refers to the output of an [XML 1.0] or [XML 1.1] processor, expressed as a collection of information items and properties as defined by the [XML Information Set] specification. In this document the term *infoset* is used as a synonym for *information set*.

The term *fatal error* refers to the presence of factors that prevent normal processing from continuing. The term *resource error* refers to a failure of an attempt to fetch a resource from a URL. XInclude processors must stop processing when encountering errors other than resource errors, which must be handled as described in § 4.4 – Fallback Behavior on page 9.

3. Syntax

XInclude defines a namespace associated with the URI `http://www.w3.org/2001/XInclude`. The XInclude namespace contains two elements with the local names `include` and `fallback`. For convenience, within this specification these elements are referred to as `xi:include` and `xi:fallback` respectively.

The following (non-normative) XML schema [XML Schemas] illustrates the content model of the `xi` namespace:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xi="http://www.w3.org/2001/XInclude"
           targetNamespace="http://www.w3.org/2001/XInclude"
           finalDefault="extension">

  <xs:element name="include" type="xi:includeType" />

  <xs:complexType name="includeType" mixed="true">
    <xs:choice minOccurs='0' maxOccurs='unbounded' >
      <xs:element ref='xi:fallback' />
      <xs:any namespace='##other' processContents='lax' />
      <xs:any namespace='##local' processContents='lax' />
    </xs:choice>
    <xs:attribute name="href" use="optional" type="xs:anyURI"/>
    <xs:attribute name="parse" use="optional" default="xml"
                  type="xi:parseType" />
    <xs:attribute name="xpointer" use="optional" type="xs:string"/>
    <xs:attribute name="encoding" use="optional" type="xs:string"/>
    <xs:attribute name="accept" use="optional" type="xs:string"/>
    <xs:attribute name="accept-language" use="optional" type="xs:string"/>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>

  <xs:simpleType name="parseType">
    <xs:restriction base="xs:token">
      <xs:enumeration value="xml"/>
      <xs:enumeration value="text"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="fallback" type="xi:fallbackType" />

  <xs:complexType name="fallbackType" mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="xi:include"/>
      <xs:any namespace="##other" processContents="lax"/>
      <xs:any namespace="##local" processContents="lax"/>
    </xs:choice>
    <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:complexType>

</xs:schema>
```

3.1. xi:include Element

The `xi:include` element has the following attributes:

href

A value which, after appropriate escaping (see § 4.1.1 – Escaping of href attribute values on page 6) has been performed, results in a URI reference or an IRI reference specifying the location of the resource to include. The href attribute is optional; the absence of this attribute is the same as specifying href="", that is, the reference is to the same document. If the href attribute is absent when parse="xml", the xpointer attribute must be present. Fragment identifiers must not be used; their appearance is a fatal error. A value that results in a syntactically invalid URI or IRI should be reported as a fatal error, but some implementations may find it impractical to distinguish this case from a resource error.

☞ A URI ending in # is considered by [IETF RFC 2396] to have an empty fragment identifier. Such a URI would result in a fatal error as described above.

☞ A key feature of XInclude is that it allows a resource to be cast to a user-specifed type for inclusion (XML or text). The returned media type is therefore essentially ignored for the purposes of inclusion processing, and the syntax of the fragment identifier of the returned media type will generally not be applicable to the user-specified type. For parse="xml" inclusions, sub-resources are identified by a separate xpointer attribute, which is applied after the casting takes place. While this does not prevent subresources of XML documents to be identified by URI (See Architecture of the World Wide Web [Identification]), it does preclude the use of these identifiers directly within XInclude.

parse

Indicates whether to include the resource as parsed XML or as text. The parse attribute allows XInclude to give the author of the including document priority over the server of the included document in terms of how to process the included content. A value of "xml" indicates that the resource must be parsed as XML and the infosets merged. A value of "text" indicates that the resource must be included as the character information items. This attribute is optional. When omitted, the value of "xml" is implied (even in the absence of a default value declaration). Values other than "xml" and "text" are a fatal error.

☞ For interoperability between validating and non-validating systems, whitespace should not appear in the parse attribute.

xpointer

When parse="xml", the XPointer (see [XPointer Framework]) contained in the xpointer attribute is evaluated to identify a portion of the resource to include. This attribute is optional; when omitted, the entire resource is included. The xpointer attribute must not be present when parse="text". If the xpointer attribute is absent, the href attribute must be present.

☞ Since the xpointer attribute is not a URI reference, and %-escaping is not done in XPointers, '%' is an ordinary character in the value of the xpointer attribute.

encoding

When parse="text", it is sometimes impossible to correctly detect the encoding of the text resource. The encoding attribute specifies how the resource is to be translated. The value of this attribute should be a valid encoding name. The encoding attribute has no effect when parse="xml".

accept

The value of the accept attribute may be used by the XInclude processor to aid in content negotiation. When the XInclude processor fetches a resource via HTTP, it should place the value of the accept attribute, if one exists, in the HTTP request as an `Accept` header as described in section 14.1 of [IETF RFC 2616]. Values containing characters outside the range #x20 through #x7E must be flagged as fatal errors.

accept-language

The value of the accept-language attribute may be used by the XInclude processor to aid in content negotiation. When the XInclude processor fetches a resource via HTTP, it should place the value of the accept-language attribute, if one exists, in the HTTP request as an `Accept-Language` header as described in section 14.4 of [IETF RFC 2616]. Values containing characters outside the range #x20 through #x7E are disallowed in HTTP headers, and must be flagged as fatal errors.

Attributes other than those listed above may be placed on the `xi:include` element. Unprefixed attribute names are reserved for future versions of this specification, and must be ignored by XInclude 1.0 processors.

The *children* property of the `xi:include` element may include a single `xi:fallback` element; the appearance of more than one `xi:fallback` element, an `xi:include` element, or any other element from the XInclude namespace is a fatal error. Other content (text, processing instructions, comments, elements not in the XInclude namespace, descendants of child elements) is not constrained by this specification and is ignored by the XInclude processor, that is, it has no effect on include processing, and does not appear in the *children* properties of the result infoset. Such content might be used by applications analyzing a pre-inclusion infoset, or be made available to an application post-inclusion through means other than the normal infoset properties.

The following (non-normative) DTD fragment illustrates a sample declaration for the `xi:include` element:

```
<!ELEMENT xi:include (xi:fallback?)>
<!ATTLIST xi:include
    xmlns:xi        CDATA       #FIXED    "http://www.w3.org/2001/XInclude"
    href            CDATA       #IMPLIED
    parse           (xml|text)  "xml"
    xpointer        CDATA       #IMPLIED
    encoding        CDATA       #IMPLIED
    accept          CDATA       #IMPLIED
    accept-language CDATA       #IMPLIED
>
```

3.2. xi:fallback Element

The `xi:fallback` element appears as a child of an `xi:include` element. It provides a mechanism for recovering from missing resources. When a resource error is encountered, the `xi:include` element is replaced with the contents of the `xi:fallback` element. If the `xi:fallback` element is empty, the `xi:include` element is removed from the result. If the `xi:fallback` element is missing, a resource error results in a fatal error.

The `xi:fallback` element can appear only as a child of an `xi:include` element. It is a fatal error for an `xi:fallback` element to appear in a document anywhere other than as the direct child of the `xi:include` (before inclusion processing on the contents of the element.) It is a fatal error for the `xi:fallback` element to contain any elements from the XInclude namespace other than `xi:include`.

Attributes may be placed on the `xi:fallback` element. Unprefixed attribute names are reserved for future versions of this specification, and must be ignored by XInclude 1.0 processors.

The content of `xi:fallback` elements are ignored unless a resource error occurs while processing the surrounding `xi:include` element. In particular, apparent fatal errors caused by the presence, absence, or content of elements and attributes inside the `xi:fallback` element must not be reported in `xi:fallback` elements that are ignored.

The following (non-normative) DTD fragment illustrates a sample declaration for the `xi:fallback` element:

```
<!ELEMENT xi:fallback ANY>
<!ATTLIST xi:fallback
```

```
    xmlns:xi    CDATA    #FIXED    "http://www.w3.org/2001/XInclude"
>
```

4. Processing Model

Inclusion as defined in this document is a specific type of [XML Information Set] transformation.

The input for the inclusion transformation consists of a *source infoset*. The output, called the *result infoset*, is a new infoset which merges the source infoset with the infosets of resources identified by URI references or IRI references appearing in `xi:include` elements. Thus a mechanism to resolve URIs or IRIs and return the identified resources as infosets is assumed. Well-formed XML entities that do not have defined infosets (e.g. an external entity with multiple top-level elements) are outside the scope of this specification, either for use as a source infoset or the result infoset.

`xi:include` elements in the source infoset serve as inclusion transformation instructions. The information items located by the `xi:include` element are called the *top-level included items* . The top-level included items together with their attributes, namespaces, and descendants, are called the *included items* . The result infoset is essentially a copy of the source infoset, with each `xi:include` element and its descendants replaced by its corresponding included items.

4.1. The Include Location

The value of the href attribute, after escaping according to § 4.1.1 – Escaping of href attribute values on page 6, is interpreted as either a URI reference or an IRI reference. The base URI for relative URIs or IRIs is the base URI of the `xi:include` element as specified in [XML Base]. The URI or IRI resulting from resolution of the normalized value of the `href` attribute (or the empty string if no attribute appears) to absolute URI or IRI form is called the *include location*.

The absence of a value for the href attribute, either by the appearance of href="" or by the absence of the href attribute, represents a case which may be incompatible with certain implementation strategies. For instance, an XInclude processor might not have a textual representation of the source infoset to include as parse="text", or it may be unable to access another part of the document using parse="xml" and an xpointer because of streamability concerns. An implementation may choose to treat any or all absences of a value for the href attribute as resource errors. Implementations should document the conditions under which such resource errors occur.

4.1.1. Escaping of href attribute values

The value of this attribute is an XML resource identifer as defined in [XML 1.1] section 4.2.2 "External Entities", which is interpreted as an IRI Reference as defined in RFC 3987 [IETF RFC 3987], after the escaping procedure described in [XML 1.1] section 4.2.2 is applied. If necessary for the implementation, the value may be further converted to a URI reference as described in [XML 1.1].

4.1.2. Using XInclude with Content Negotiation

The use of a mechanism like HTTP [IETF RFC 2616] content negotiation introduces an additional level of potential complexity into the use of XInclude. Developers who use XInclude in situations where content negotiation is likely or possible should be aware of the possibility that they will be including content that may differ structurally from the content they expected, even if that content is XML. For example, a single URI or IRI may variously return a raw XML representation of the resource, an XSL-FO [XSL-FO] representation, or an XHTML [XHTML] representation, as well as versions in different character encodings or languages.

Authors whose XInclude processing depends on the receipt of a particular vocabulary of XML should use the accept and accept-language attributes to increase the probability that the resource is provided in the expected format.

4.2. Included Items when parse="xml"

When parse="xml", the include location is dereferenced, the resource is fetched, and an infoset is created by parsing the resource as if the media type were application/xml (including character encoding determination).

The specifics of how an infoset is created are intentionally unspecified, to allow for flexibility by implementations and to avoid defining a particular processing model for components of the XML architecture. Particulars of whether DTD or XML schema validation are performed, for example, are not constrained by this specification.

The character encodings of the including and included resources can be different. This does not affect the resulting infoset, but might need to be taken into account during any subsequent serialization.

Resources that are unavailable for any reason (for example the resource doesn't exist, connection difficulties or security restrictions prevent it from being fetched, the URI scheme isn't a fetchable one, the resource is in an unsupported encoding, or the resource is determined through implementation-specific mechanisms not to be XML) result in a resource error. Resources that contain non-well-formed XML result in a fatal error.

The distinction between a resource error and a fatal error is somewhat implementation-dependent. Consider an include location returning an HTML document, perhaps as an error page. One processor might determine that no infoset can be created from the resource (by examining the media type, for example) and raise a resource error, enabling fallback behavior. Another processor with no such heuristics might attempt to parse the non-XML resource as XML and encounter a well-formedness (fatal) error.

`xi:include` elements in this infoset are recursively processed to create the *acquired infoset*. For an intra-document reference (via xpointer attribute) the source infoset is used as the acquired infoset.

The portion of the acquired infoset to be included is called the *inclusion target*. The *document information item* of the acquired infoset serves as the inclusion target unless the xpointer attribute is present and identifies a subresource. XPointers of the forms described in [XPointer Framework] and [XPointer element() scheme] must be supported. XInclude processors optionally support other forms of XPointer such as that described in [XPointer xpointer() Scheme]. An error in the XPointer is a resource error.

The [XPointer xpointer() Scheme] is not specified in terms of the [XML Information Set], but instead is based on the [XPath 1.0] Data Model, because the XML Information Set had not yet been developed. The mapping between XPath node locations and information items is straightforward. However, xpointer() assumes that all entities have been expanded. Thus it is a fatal error to attempt to resolve an xpointer() scheme on a document that contains *unexpanded entity reference information items*.

The set of top-level included items is derived from the acquired infoset as follows.

4.2.1. Document Information Items

The inclusion target might be a *document information item* (for instance, no specified xpointer attribute, or an XPointer specifically locating the document root.) In this case, the set of top-level included items is the *children* of the acquired infoset's document information item, except for the *document type declaration information item* child, if one exists.

☞ The XML Information Set specification does not provide for preservation of white space outside the document element. XInclude makes no further provision to preserve this white space.

4.2.2. Multiple Nodes

The inclusion target might consist of more than a single node. In this case the set of top-level included items is the set of information items from the acquired infoset corresponding to the nodes referred to by the XPointer, in the order in which they appear in the acquired infoset.

4.2.3. Range Locations

The inclusion target might be a location set that represents a range or a set of ranges.

Each range corresponds to a set of information items in the acquired infoset. An information item is said to be *selected* by a range if it occurs after (in document order) the starting point of the range and before the ending point of the range. An information item is said to be *partially selected* by a range if it contains only the starting point of the range, or only the ending point of the range. By definition, a character information item cannot be partially selected.

The set of top-level included items is the union, in document order with duplicates removed, of the information items either selected or partially selected by the range. The *children* property of selected information items is not modified. The *children* property of partially selected information items is the set of information items that are in turn either selected or partially selected, and so on.

4.2.4. Point Locations

The inclusion target might be a location set that represents a point. In this case the set of included items is empty.

4.2.5. Element, Comment, and Processing Instruction Information Items

The inclusion target might be an element node, a comment node, or a processing instruction node, respectively representing an *element information item*, a *comment information item*, or a *processing instruction information item*. In this case the set of top-level included items consists of the information item corresponding to the element, comment, or processing instruction node in the acquired infoset.

4.2.6. Attribute and Namespace Declaration Information Items

It is a fatal error for the inclusion target to be an attribute node or a namespace node.

4.2.7. Inclusion Loops

When recursively processing an `xi:include` element, it is a fatal error to process another `xi:include` element with an include location and xpointer attribute value that have already been processed in the inclusion chain.

In other words, the following are all legal:

- An `xi:include` element may reference the document containing the include element, when parse="text".
- An `xi:include` element may identify a different part of the same local resource (same href, different xpointer).
- Two non-nested `xi:include` elements may identify a resource which itself contains an `xi:include` element.

The following are illegal:

- An `xi:include` element pointing to itself or any ancestor thereof, when parse="xml".
- An `xi:include` element pointing to any include element or ancestor thereof which has already been processed at a higher level.

4.3. Included Items when parse="text"

When parse="text", the include location is dereferenced and the resource is fetched and transformed to a set of character information items. This feature facilitates the inclusion of working XML examples, as well as other text-based formats.

Resources that are unavailable for any reason (for example the resource doesn't exist, connection difficulties or security restrictions prevent it from being fetched, the URI scheme isn't a fetchable one, or the resource is in an unsupported encoding) result in a resource error.

The encoding of such a resource is determined by:

- external encoding information, if available, otherwise
- if the media type of the resource indicates, according to XML Media Types [IETF RFC 3023], that the resource is XML, for example `text/xml` or `application/xml` or matches `text/*+xml` or `application/*+xml`, then the encoding is determined as specified in [XML 1.0] or [XML 1.1] section 4.3.3, as appropriate, otherwise
- the value of the encoding attribute if one exists, otherwise
- UTF-8.

Byte sequences outside the range allowed by the encoding are a fatal error. Characters that are not permitted in XML documents also are a fatal error.

Each character obtained from the transformation of the resource is represented in the top-level included items as a *character information item* with the *character code* set to the character code in ISO 10646 encoding, and the *element content whitespace* set to false.

When the first character is U+FEFF and is interpreted as a Byte-Order Mark, it should be discarded. It is interpreted as a BOM in UTF-8, UTF-16, and UTF-32 encodings; it is not interpreted as a BOM in the UTF-16LE, UTF-16BE, UTF-32LE, and UTF-32BE encodings.

The [Character Model] discusses normalization of included text.

4.4. Fallback Behavior

XInclude processors must perform fallback behavior in the event of a resource error, as follows:

If the *children* of the `xi:include` element information item in the source infoset contain exactly one `xi:fallback` element, the top-level included items consist of the information items corresponding to the result of performing XInclude processing on the *children* of the `xi:fallback` element. It is a fatal error if there is zero or more than one `xi:fallback` element.

☞ Fallback content is not dependent on the value of the `parse` attribute. The `xi:fallback` element can contain markup even when `parse="text"`. Likewise, it can contain a simple string when `parse="xml"`.

4.5. Creating the Result Infoset

The result infoset is a copy of the source infoset, with each `xi:include` element processed as follows:

The information item for the `xi:include` element is found. The *parent* property of this item refers to an information item called the *include parent*. The *children* property of the include parent is modified by replacing the `xi:include` element information item with the top-level included items. The *parent* property of each included item is set to the include parent.

It is a fatal error to attempt to replace an `xi:include` element appearing as the document (top-level) element in the source infoset with something other than a list of zero or more comments, zero or more processing instructions, and one element.

Some processors may not be able to represent an element's *in-scope namespaces* property if it does not include bindings for all the prefixes bound in its parent's *in-scope namespaces*. Such processors may therefore include additional namespace bindings inherited from the include parent in the *in-scope namespaces* of the included items.

The inclusion history of each top-level included item is recorded in the extension property *include history*. The *include history* property is a list of *element information items*, representing the `xi:include` elements for recursive levels of inclusion. If an *include history* property already appears on a top-level included item, the `xi:include` element information item is prepended to the list. If no *include history* property exists, then this property is added with the single value of the `xi:include` element information item.

The included items will all appear in the result infoset. This includes *unexpanded entity reference information items* if they are present.

Intra-document references within `xi:include` elements are resolved against the source infoset. The effect of this is that the order in which `xi:include` elements are processed does not affect the result.

In the following example, the second include always points to the first `xi:include` element and not to itself, regardless of the order in which the includes are processed. Thus the result of this inclusion is two copies of `something.xml`, and does not produce an inclusion loop error.

```
<x xmlns:xi="http://www.w3.org/2001/XInclude">
  <xi:include href="something.xml"/>
  <xi:include
xpointer="xmlns(xi=http://www.w3.org/2001/XInclude)xpointer(x/xi:include[1])"
              parse="xml"/>
</x>
```

An XInclude processor may, at user option, suppress `xml:base` and/or `xml:lang` fixup.

4.5.1. Unparsed Entities

Any *unparsed entity information item* appearing in the *references* property of an attribute on the included items or any descendant thereof is added to the *unparsed entities* property of the result infoset's *document information item*, if it is not a duplicate of an existing member. Duplicates do not appear in the result infoset.

Unparsed entity items with the same *name*, *system identifier*, *public identifier*, *declaration base URI*, *notation name*, and *notation* are considered to be duplicate. An application may also be able to detect that unparsed entities are duplicate through other means. For instance, the URI resulting from combining the system identifier and the declaration base URI is the same.

It is a fatal error to include unparsed entity items with the same name, but which are not determined to be duplicates.

4.5.2. Notations

Any *notation information item* appearing in the *references* property of an attribute in the included items or any descendant thereof is added to the *notations* property of the result infoset's *document information item*, if it is not a duplicate of an existing member. Likewise, any notation referenced by an unparsed entity added as described in § 4.5.1 – Unparsed Entities on page 11, is added unless it is a duplicate. Duplicates do not appear in the result infoset.

Notation items with the same *name*, *system identifier*, *public identifier*, and *declaration base URI* are considered to be duplicate. An application may also be able to detect that notations are duplicate through other means. For instance, the URI resulting from combining the system identifier and the declaration base URI is the same.

It is a fatal error to include notation items with the same name, but which are not determined to be duplicates.

4.5.3. *references* Property Fixup

During inclusion, an *attribute information item* whose *attribute type* property is IDREF or IDREFS has a *references* property with zero or more element values from the source or included infosets. These values must be adjusted to correspond to element values that occur in the result infoset. During this process, XInclude also corrects inconsistencies between the *references* property and the *attribute type* property, which might arise in the following circumstances:

- A document fragment contains an IDREF pointing to an element in the included document but outside the part being included. In this case there is no element in the result infoset that corresponds to the element value in the original *references* property.
- A document or document fragment is not self-contained. That is, it contains IDREFs which do not refer to an element within that document or document fragment, with the intention that these references will be realized after inclusion. In this case, the value of the *references* property is unknown or has no value.
- The result infoset has ID clashes - that is, more than one attribute with *attribute type* ID with the same *normalized value*. In this case, attributes with *attribute type* IDREF or IDREFS with the same *normalized value* might have different values for their *references* properties.

In resolving these inconsistencies, XInclude takes the *attribute type* property as definitive. In the result infoset, the value of the *references* property of an *attribute information item* whose *attribute type* property is IDREF or IDREFS is adjusted as follows:

For each token in the *normalized value* property, the *references* property contains an *element information item* with the same properties as the *element information item* in the result infoset with an attribute with *attribute type* ID and *normalized value* equal to the token. The order of the elements in the *references* property is the same as the order of the tokens appearing in the *normalize value*. If for any of the token values, no element or more than one element is found, the *references* property has no value.

4.5.4. Namespace Fixup

The *in-scope namespaces* property ensures that namespace scope is preserved through inclusion. However, after inclusion, the *namespace attributes* property might not provide the full list of namespace declarations necessary to interpret qualified names in attribute or element content in the result. It is therefore not recommended that XInclude processors expose *namespace attributes* in the result. If this is unavoidable, the implementation may add *attribute information items* to the *namespace attributes* property in order to approximate the information conveyed by *in-scope namespaces*.

4.5.5. Base URI Fixup

The base URI property of the acquired infoset is not changed as a result of merging the infoset, and remains unchanged after merging. Thus relative URI references in the included infoset resolve to the same URI despite being included into a document with a potentially different base URI in effect. xml:base attributes are added to the result infoset to indicate this fact.

Each *element information item* in the top-level included items which has a different *base URI* than its include parent has an *attribute information item* added to its *attributes* property. This attribute has the following properties:

1. A *namespace name* of `http://www.w3.org/XML/1998/namespace`.
2. A *local name* of `base`.
3. A *prefix* of `xml`.
4. A *normalized value* equal to either the *base URI* of the element, or an equivalent URI reference relative to the *base URI* of the include parent. The circumstances in which a relative URI is desirable, and how to compute such a relative URI, are implementation-dependent.
5. A *specified* flag indicating that this attribute was actually specified in the start-tag of its element.
6. An *attribute type* of `CDATA`.
7. A *references* property with no value.
8. An *owner element* of the information item of the element.

If an xml:base attribute information item is already present, it is replaced by the new attribute.

4.5.6. Language Fixup

While the `xml:lang` attribute is described as inherited by XML, the XML Information Set makes no provision for preserving the inheritance of this property through document composition such as XInclude provides. This section introduces a *language* property which records the scope of `xml:lang` information in order to preserve it during inclusion.

An XInclude processor should augment the source infoset and the acquired infoset by adding the *language* property to each *element information item*. The value of this property is the *normalized value* of the `xml:lang` attribute appearing on that element if one exists, with xml:lang="" resulting in no value, otherwise it is the value of the *language* property of the element's parent element if one exists, otherwise the property has no value.

Each *element information item* in the top-level included items which has a different value of *language* than its include parent (taking case-insensitivity into account per [IETF RFC 3066]), or that has a value if its include parent is a *document information item*, has an *attribute information item* added to its *attributes* property. This attribute has the following properties:

1. A *namespace name* of `http://www.w3.org/XML/1998/namespace`.
2. A *local name* of `lang`.
3. A *prefix* of `xml`.
4. A *normalized value* equal to the *language* property of the element. If the *language* property has no value, the normalized value is the empty string.
5. A *specified* flag indicating that this attribute was actually specified in the start-tag of its element.
6. An *attribute type* of `CDATA`.
7. A *references* property with no value.
8. An *owner element* of the information item of the element.

If an xml:lang attribute information item is already present, it is replaced by the new attribute.

☞ The `xml:space` attribute is not treated specially by XInclude.

4.5.7. Properties Preserved by the Infoset

As an infoset transformation, XInclude operates on the logical structure of XML documents, not on their text serialization. All properties of an information item described in [XML Information Set] other than those specifically modified by this specification are preserved during inclusion. The *include history* and *language* properties introduced in this specification is also preserved. Extension properties such as [XML Schemas] Post Schema Validation Infoset (PSVI) properties are discarded by default. However, an XInclude processor may, at user option, preserve these properties in the resulting infoset if they are correct according to the specification describing the semantics of the extension properties.

For instance, the PSVI *validity* property describes the conditions of ancestors and descendants. Modification of ancestors and descendants during the XInclude process can render the value of this property inaccurate. By default, XInclude strips this property, but by user option the property could be recalculated to obtain a semantically accurate value. Precisely how this is accomplished is outside the scope of this specification.

5. Conformance

5.1. Markup Conformance

An *element information item* conforms to this specification if it meets the structural requirements for include elements defined in this specification. This specification imposes no particular constraints on DTDs or XML schemas; conformance applies only to elements and attributes.

5.2. Application Conformance

An application conforms to XInclude if it:

- supports [XML 1.0] and [Namespaces in XML] or [XML 1.1] and [Namespaces in XML 1.1], the [XML Information Set], [XML Base], the [XPointer Framework], and the [XPointer element() scheme];
- stops processing when a fatal error is encountered;
- observes the mandatory conditions (must) set forth in this specification, and for any optional conditions (should and may) it chooses to observe, observes them in the way prescribed; and
- performs markup conformance testing according to all the conformance constraints appearing in this specification.

Support for the [XPointer xpointer() Scheme] is not mandatory for full XInclude conformance. Authors are advised that use of xpointer() and other XPointer schemes than element() might not be supported by all conformant XInclude implementations.

5.3. XML Information Set Conformance

This specification conforms to the [XML Information Set]. The following information items must be present in the input infosets to enable correct processing:

- *Document information items* with *children* and *base URI* properties.
- *Element information items* with *namespace name*, *local name*, *children*, *attributes*, *base URI* and *parent* properties.
- *Attribute information items* with *namespace name*, *local name* and *normalized value* properties.

Additionally, XInclude processing might generate the following kinds of information items in the result:

- *Character information items* with *character code*, *element content whitespace* and *parent* properties.

XInclude extends the infoset with the property *include history*, which may appear on the following types of information items in the result:

- *Element information items*.
- *Processing instruction information items*.
- *Comment information items*.
- *Character information items*.

XInclude also extends the infoset with the property *language*, which may appear on *element information items* in the result.

Appendix A. References

IETF RFC 2119

RFC 2119: Key words for use in RFCs to Indicate Requirement Levels. Internet Engineering Task Force, 1997. Available at `http://www.ietf.org/rfc/rfc2119.txt`.

IETF RFC 3629

RFC 3629: UTF-8, a transformation format of ISO 10646. Internet Engineering Task Force, 2003. Available at `http://www.ietf.org/rfc/rfc3629.txt`.

IETF RFC 2396

RFC 2396: Uniform Resource Identifiers. Internet Engineering Task Force, 1995. Available at `http://www.ietf.org/rfc/rfc2396.txt`.

IETF RFC 2732

RFC 2732: Format for Literal IPv6 Addresses in URL's. Internet Engineering Task Force, 1999. Available at `http://www.ietf.org/rfc/rfc2732.txt`.

IETF RFC 3023

RFC 3023: XML Media Types. Internet Engineering Task Force, 2001. Available at `http://www.ietf.org/rfc/rfc3023.txt`.

IETF RFC 3987

RFC 3987: Internationalized Resource Identifiers (IRIs). Internet Engineering Task Force, 2005. Available at `http://www.ietf.org/rfc/rfc3987.txt`.

Unicode

The Unicode Consortium. *The Unicode Standard, Version 4.0.* Reading, Mass.: Addison-Wesley, 2003, as updated from time to time by the publication of new versions. (See http://www.unicode.org/unicode/standard/versions/ for the latest version and additional information on versions of the standard and of the Unicode Character Database).

XML 1.0

Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, Eve Maler, François Yergeau, editors. *Extensible Markup Language (XML) 1.0 (Third Edition)*, World Wide Web Consortium, 2004. Available at `http://www.w3.org/TR/2004/REC-xml-20040204/`.

XML 1.1

Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, Eve Maler, François Yergeau, John Cowan, editors. *Extensible Markup Language (XML) 1.1*, World Wide Web Consortium, 2004. Available at `http://www.w3.org/TR/2004/REC-xml11-20040204/`.

XML Base

Jonathan Marsh, editor. *XML Base*. World Wide Web Consortium, 2001. Available at `http://www.w3.org/TR/2001/REC-xmlbase-20010627/`.

XML Information Set

John Cowan and Richard Tobin, editors. *XML Information Set (Second Edition)*. World Wide Web Consortium, 2004. Available at `http://www.w3.org/TR/2004/REC-xml-infoset-20040204/`.

Namespaces in XML

Tim Bray, Dave Hollander, and Andrew Layman, editors. *Namespaces in XML*. World Wide Web Consortium, 1999. Available at `http://www.w3.org/TR/1999/REC-xml-names-19990114/`.

Namespaces in XML 1.1

Tim Bray, Dave Hollander, Andrew Layman, Richard Tobin, editors. *Namespaces in XML 1.1*. World Wide Web Consortium, 2004. Available at `http://www.w3.org/TR/2004/REC-xml-names11-20040204/`.

XPointer Framework

Paul Grosso, Eve Maler, Jonathan Marsh, Norman Walsh, editors. *XPointer Framework*. World Wide Web Consortium, 2003. Available at `http://www.w3.org/TR/2003/REC-xptr-framework-20030325/`.

XPointer element() scheme

Paul Grosso, Eve Maler, Jonathan Marsh, Norman Walsh, editors. *XPointer element() Scheme*. World Wide Web Consortium, 2003. Available at `http://www.w3.org/TR/2003/REC-xptr-element-20030325/`.

Appendix B. References (Non-Normative)

IETF RFC 2616

RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1. Internet Engineering Task Force, 1999. Available at `http://www.ietf.org/rfc/rfc2616.txt`.

IETF RFC 3066

RFC 3066: Tags for the Identification of Languages. Internet Engineering Task Force, 2001. Available at `http://www.ietf.org/rfc/rfc3066.txt`.

XML Inclusion Proposal

Jonathan Marsh, David Orchard, editors. *XML Inclusion Proposal (XInclude)*. World Wide Web Consortium, 2004. Available at `http://www.w3.org/TR/1999/NOTE-xinclude-19991123`.

XML Linking Language

Steve DeRose, Eve Maler, David Orchard, and Ben Trafford, editors. *XML Linking Language (XLink)*. World Wide Web Consortium, 2001. Available at `http://www.w3.org/TR/2001/REC-xlink-20010627/`.

XPointer xpointer() Scheme

Steve DeRose, Ron Daniel, Eve Maler, editors. *XPointer xpointer() Scheme*. World Wide Web Consortium, 2002. Available at `http://www.w3.org/TR/2002/WD-xptr-xpointer-20021219/`.

XPath 1.0

James Clark, Steve DeRose, editors. *XML Path Language (XPath) Version 1.0*. World Wide Web Consortium, 1999. Available at `http://www.w3.org/TR/1999/REC-xpath-19991116`.

Character Model

Martin J. Dürst, François Yergeau, Misha Wolf, Asmus Freytag, Tex Texin. *Character Model for the World Wide Web 1.0: Normalization*. World Wide Web Consortium, 2001. Available at `http://www.w3.org/TR/charmod-norm/`.

XML Schemas

Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, editors. *XML Schema Part 1: Structures*. World Wide Web Consortium, 2001. Available at `http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/`.

XSL-FO

Sharon Adler *et al*. *Extensible Stylesheet Language (XSL)*. World Wide Web Consortium, 2001. Available at `http://www.w3.org/TR/2001/REC-xsl-20011015/`.

XHTML

Steven Pemberton *et al*. *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)*. World Wide Web Consortium, 2002. Available at `http://www.w3.org/TR/2002/REC-xhtml1-20020801/`.

Appendix C. Examples (Non-Normative)

C.1. Basic Inclusion Example

The following XML document contains an `xi:include` element which points to an external document. Assume the base URI of this document is `http://www.example.org/document.xml`.

```
<?xml version='1.0'?>
<document xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>120 Mz is adequate for an average home user.</p>
  <xi:include href="disclaimer.xml"/>
</document>
```

disclaimer.xml contains:

```
<?xml version='1.0'?>
<disclaimer>
  <p>The opinions represented herein represent those of the individual
  and should not be interpreted as official policy endorsed by this
  organization.</p>
</disclaimer>
```

The infoset resulting from resolving inclusions on this document is the same (except for the *include history* and *language* properties) as that of the following document:

```
<?xml version='1.0'?>
<document xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>120 Mz is adequate for an average home user.</p>
  <disclaimer xml:base="http://www.example.org/disclaimer.xml">
  <p>The opinions represented herein represent those of the individual
  and should not be interpreted as official policy endorsed by this
  organization.</p>
</disclaimer>
</document>
```

C.2. Textual Inclusion Example

The following XML document includes a "working example" into a document.

```
<?xml version='1.0'?>
<document xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>This document has been accessed
  <xi:include href="count.txt" parse="text"/> times.</p>
</document>
```

where count.txt contains:

```
324387
```

The infoset resulting from resolving inclusions on this document is the same (except for the *include history* and *language* properties) as that of the following document:

```
<?xml version='1.0'?>
<document xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>This document has been accessed
  324387 times.</p>
</document>
```

C.3. Textual Inclusion of XML Example

The following XML document includes a "working example" into a document.

```
<?xml version='1.0'?>
<document xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>The following is the source of the "data.xml" resource:</p>
  <example><xi:include href="data.xml" parse="text"/></example>
</document>
```

data.xml contains:

```
<?xml version='1.0'?>
<data>
  <item><![CDATA[Brooks & Shields]]></item>
</data>
```

The infoset resulting from resolving inclusions on this document is the same (except for the *include history* and *language* properties) as that of the following document:

```
<?xml version='1.0'?>
<document xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>The following is the source of the "data.xml" resource:</p>
  <example>&lt;?xml version='1.0'?&gt;
&lt;data&gt;
  &lt;item&gt;&lt;![CDATA[Brooks &amp; Shields]]&gt;&lt;/item&gt;
&lt;/data&gt;</example>
</document>
```

C.4. Fragment Inclusion Example

The following illustrates the results of including fragments of another XML document. Assume the base URI of the document is `http://www.example.com/JoeSmithQuote.xml`.

```
<?xml version='1.0'?>
<price-quote xmlns:xi="http://www.w3.org/2001/XInclude">
  <prepared-for>Joe Smith</prepared-for>
  <good-through>20040930</good-through>
  <xi:include href="price-list.xml" xpointer="w002-description"/>
  <volume>40</volume>
  <xi:include href="price-list.xml" xpointer="element(w002-prices/2)"/>
</price-quote>
```

price-list.xml references a DTD which declares the id attributes as type `ID`, and contains:

```
<?xml version='1.0'?>
<!DOCTYPE price-list SYSTEM "price-list.dtd">
<price-list xml:lang="en-us">
  <item id="w001">
    <description id="w001-description">
      <p>Normal Widget</p>
    </description>
    <prices id="w001-prices">
      <price currency="USD" volume="1+">39.95</price>
      <price currency="USD" volume="10+">34.95</price>
      <price currency="USD" volume="100+">29.95</price>
    </prices>
  </item>
  <item id="w002">
    <description id="w002-description">
      <p>Super-sized widget with bells <i>and</i> whistles.</p>
    </description>
    <prices id="w002-prices">
      <price currency="USD" volume="1+">59.95</price>
      <price currency="USD" volume="10+">54.95</price>
      <price currency="USD" volume="100+">49.95</price>
    </prices>
  </item>
</price-list>
```

The infoset resulting from resolving inclusions on this document is the same (except for the *include history* and *language* properties) as that of the following document:

```
<?xml version='1.0'?>
<price-quote xmlns:xi="http://www.w3.org/2001/XInclude">
  <prepared-for>Joe Smith</prepared-for>
  <good-through>20040930</good-through>
  <description id="w002-description" xml:lang="en-us"
```

```
               xml:base="http://www.example.com/price-list.xml">
    <p>Super-sized widget with bells <i>and</i> whistles.</p>
  </description>
  <volume>40</volume>
  <price currency="USD" volume="10+" xml:lang="en-us"
         xml:base="http://www.example.com/price-list.xml">54.95</price>
</price-quote>
```

C.5. Range Inclusion Example

The following illustrates the results of including a range specified by an XPointer. Assume the base URI of the document is `http://www.example.com/document.xml`.

```
<?xml version='1.0'?>
<document>
  <p>The relevant excerpt is:</p>
  <quotation>
    <include xmlns="http://www.w3.org/2001/XInclude"
      href="source.xml" xpointer="xpointer(string-range(chapter/p[1],'Sentence 2')/

             range-to(string-range(/chapter/p[2]/i,'3.',1,2)))"/>
  </quotation>
</document>
```

source.xml contains:

```
<chapter>
  <p>Sentence 1.  Sentence 2.</p>
  <p><i>Sentence 3.  Sentence 4.</i>  Sentence 5.</p>
</chapter>
```

The infoset resulting from resolving inclusions on this document is the same (except for the *include history* and *language* properties) as that of the following document:

```
<?xml version='1.0'?>
<document>
  <p>The relevant excerpt is:</p>
  <quotation>
    <p xml:base="http://www.example.com/source.xml">Sentence 2.</p>
  <p xml:base="http://www.example.com/source.xml"><i>Sentence 3.</i></p>
  </quotation>
</document>
```

C.6. Fallback Example

The following XML document relies on the fallback mechanism to succeed in the event that the resources `example.txt` and `fallback-example.txt` are not available..

```
<?xml version='1.0'?>
<div>
  <xi:include href="example.txt" parse="text"
xmlns:xi="http://www.w3.org/2001/XInclude">
    <xi:fallback><xi:include href="fallback-example.txt" parse="text">
       <xi:fallback><a href="mailto:bob@example.org">Report error</a></xi:fallback>

      </xi:include></xi:fallback>
  </xi:include>
</div>
```

If neither `example.txt` nor `fallback-example.txt` are available, the infoset resulting from resolving inclusions on this document is the same (except for the *include history* and *language* properties) as that of the following document:

```
<?xml version='1.0'?>
<div>
  <a href="mailto:bob@example.org">Report error</a>
</div>
```

This page is intentionally left blank.