XPointer Framework

W3C Recommendation 25 March 2003

This version:

http://www.w3.org/TR/2003/REC-xptr-framework-20030325/

Latest version:

http://www.w3.org/TR/xptr-framework/

Previous version:

http://www.w3.org/TR/2002/PR-xptr-framework-20021113/

Authors and Contributors:

Paul Grosso (Arbortext, Inc.) <paul@arbortext.com>
Eve Maler (Sun Microsystems) <eve.maler@sun.com>
Jonathan Marsh (Microsoft) <jmarsh@microsoft.com>
Norman Walsh (Sun Microsystems) <Norman.Walsh@Sun.COM>

Abstract

This specification defines the XML Pointer Language (XPointer) Framework, an extensible system for XML addressing that underlies additional XPointer scheme specifications. The framework is intended to be used as a basis for fragment identifiers for any resource whose Internet media type is one of `text/xml`, `application/xml`, `text/xml-external-parsed-entity`, or `application/xml-external-parsed-entity`. Other XML-based media types are also encouraged to use this framework in defining their own fragment identifier languages.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This document is a Recommendation (REC) of the W3C. It has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference 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 has been produced by the W3C XML Linking Working Group as part of the XML Activity. It is intended to address a core subset of the original XPointer requirements, and to serve, along with the accompanying XPointer element() Scheme and XPointer xmlns() Scheme specifications, as all or a foundational part of a fragment identifier syntax for the XML Media types.

Public comments on this Recommendation are welcome. Please send them to the public mailing list www-xml-linking-comments@w3.org (archive).

Information about implementations relevant to this specification and the accompanying XPointer element() Scheme and XPointer xmlns() Scheme can be found in the Implementation Report.

There are patent disclosures and license commitments associated with this Recommendation, which may be found on the XPointer IPR Statement page in conformance with W3C policy.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR/. W3C publications may be updated, replaced, or obsoleted by other documents at any time.

Table of Contents

Appendices

This page is intentionally left blank.

1. Introduction

This specification defines the XML Pointer Language (XPointer) Framework, an extensible system for XML addressing that underlies additional XPointer scheme specifications. The framework is intended to be used as a basis for fragment identifiers for any resource whose Internet media type is one of `text/xml`, `application/xml`, `text/xml-external-parsed-entity`, or `application/xml-external-parsed-entity`. Other XML-based media types are also encouraged to use this framework in defining their own fragment identifier languages.

Many types of XML-processing applications need to address into the internal structures of XML resources using URI references, for example, the XML Linking Language [XLink], XML Inclusions [XInclude], the Resource Description Framework [RDF], and SOAP 1.2 [SOAP12]. This specification does not constrain the types of applications that utilize URI references to XML resources, nor does it constrain or dictate the behavior of those applications once they locate the desired information in those resources.

1.1. Notation

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 [RFC 2119].

The formal grammar for the XPointer Framework is given using simple Extended Backus-Naur Form (EBNF) notation, as described in the XML Recommendation [XML].

1.2. Terminology

pointer

A string conforming to this specification. This specification defines the syntax and semantics of pointers.

pointer part

A portion of a pointer that provides a scheme name and some pointer data that conforms to the definition of that scheme. The XPointer processor evaluates a pointer part to identify zero or more subresources within an XML resource.

scheme

A specialized pointer data format that has a name and is defined in a specification.

XPointer processor

A software component that identifies subresources of an XML resource by applying a pointer to it. This specification defines the behavior of XPointer processors.

application

A software component that incorporates or uses an XPointer processor because it needs to access XML subresources. The occurrence and usage of XPointers, and the behavior to be applied to resources and subresources obtained by processing those XPointers, are governed by the definition of each application's corresponding data format (which could be XML-based or non-XML-based). For example, HTML [HTML] Web browsers and XInclude processors are applications that might use XPointer processors.

error

A violation of the syntactic rules of this specification, or the failure of a pointer to identify subresources.

namespace binding context

A binding of XML-namespace-defined [XML-Names] namespace prefixes to their associated namespace names.

2. Conformance

This specification defines a framework; it does not currently define a minimum conformance level for XPointer processors. Thus, the information in this section defines conformance requirements only for the framework portion of any minimum conformance level.

XPointer processors depend on the ability of applications to reverse any fragment identifier encoding and escaping (see § 4 – Character Escaping on page 6).

XPointer processor behaviour depends on the availability of certain information from an XML resource: in the terms provided by the [Infoset], the information items and properties tabulated below may be relevant. The presence of some of these items and properties depends in turn on conformant DTD or XML Schema processing: conformant XPointer processors are *not* required to do such processing, but *if* they do, shorthand pointer processing will take advantage of the information thus provided (see § 3.2 – Shorthand Pointer on page 4).

- From the XML Information Set itself [Infoset]:
  - document information item
    - `[document element]` property

      Note that if the XML resource is not a document but rather an external parsed entity, this property will not be reported. Rather, the information set is effectively extended to report the one or more top-level elements in the entity as ordered “root element” properties for the entity.
  - element information item
    - `[attributes]` property
    - `[children]` property
  - attribute information item
    - `[attribute type]` property
    - `[normalized value]` property
- From the XML Schema post-schema validation information set (PSVI) [XMLSchema], the following properties of attribute information items and element information items:
  - `[schema normalized value]` property
  - Either:
    - `[member type definition]` property
    - `[type definition]` property

- and the `[name]`, `[target namespace]` and `[base type definition]` properties of the value thereof

or:

- `[member type definition namespace]` property
- `[member type definition name]` property
- `[type definition namespace]` property
- `[type definition name]` property

Software components claiming to be XPointer processors must conform to this XPointer Framework specification and any other specifications that, together with this specification, define the minimum conformance level for XPointer, and may conform to additional XPointer scheme specifications. XPointer processors must document the additional scheme specifications to which they conform. Specifications that depend on XPointer processing should document the schemes they require and support.

Conforming XPointer processors must report XPointer Framework errors to the application. Applications are free to terminate or recover from XPointer Framework errors in any fashion.

3. Language and Processing

This section describes the XPointer Framework and the behavior of XPointer processors with respect to the framework.

An XPointer processor takes as input an XML resource and a string to be used as a pointer (for example, a fragment identifier, with escaping reversed, taken from the URI reference that was used to access the resource), attempts to evaluate the pointer with respect to the resource, and produces as output an identification of subresources, or one or more errors.

3.1. Syntax

If a string used as a pointer does not adhere to the syntax defined in this section, it is an error.

The symbol S is defined in [XML]. The symbols NCName and QName are defined in [XML-Names].

XPointer Framework Syntax

[1]	Pointer	::=	**Shorthand** | **SchemeBased**
[2]	Shorthand	::=	NCName
[3]	SchemeBased	::=	**PointerPart** (S? **PointerPart**)*
[4]	PointerPart	::=	**SchemeName** '(' **SchemeData** ')'
[5]	SchemeName	::=	QName
[6]	SchemeData	::=	**EscapedData***
[7]	EscapedData	::=	**NormalChar** | '^(' | '^)' | '^^' | '(' **SchemeData** ')'
[8]	NormalChar	::=	UnicodeChar - [()^]
[9]	UnicodeChar	::=	[#x0-#x10FFFF]

As shown in the above productions, the end of a pointer part is signaled by the right parenthesis "`)`" character that balances the left parenthesis "`(`" character that began the part. If either a left or a right parenthesis occurs in scheme data without being balanced by its counterpart, it must be escaped with a circumflex (`^`) character preceding it. Escaping pairs of balanced parentheses is allowed. Any literal occurrences of the circumflex must be escaped with an additional circumflex (that is, `^^`). Any other use of a circumflex is an error.

3.2. Shorthand Pointer

A shorthand pointer, formerly known as a barename, consists of an NCName alone. It identifies at most one element in the resource's information set; specifically, the first one (if any) in document order that has a matching NCName as an identifier. The identifiers of an element are determined as follows:

1. If an element information item has an attribute information item among its `[attributes]` that is a schema-determined ID, then it is identified by the value of that attribute information item's `[schema normalized value]` property;
2. If an element information item has an element information item among its `[children]` that is a schema-determined ID, then it is identified by the value of that element information item's `[schema normalized value]` property;
3. If an element information item has an attribute information item among its `[attributes]` that is a DTD-determined ID, then it is identified by the value of that attribute information item's `[normalized value]` property.
4. An element information item may also be identified by an externally-determined ID value.

If no element information item is identified by a shorthand pointer's NCName, the pointer is in error.

An element information item might be identified by multiple values, in a document with more than one of DTD-determined IDs, schema-determined IDs, and externally-determined IDs. In such documents, a loss of interoperability might result if the identifier values for a particular element are not always the same.

An element or attribute information item is a *schema-determined ID* if and only if one of the following is true:

1. It has a `[member type definition]` or `[type definition]` property whose value in turn has `[name]` equal to `ID` and `[target namespace]` equal to `http://www.w3.org/2001/XMLSchema`;
2. It has a `[base type definition]` whose value has that `[name]` and `[target namespace]`;
3. It has a `[base type definition]` whose value has a `[base type definition]` whose value has that `[name]` and `[target namespace]`, and so on following the `[base type definition]` property recursively;
4. It has a `[type definition name]` equal to `ID` and a `[type definition namespace]` equal to `http://www.w3.org/2001/XMLSchema`;
5. It has a `[member type definition name]` equal to `ID` and a `[member type definition namespace]` equal to `http://www.w3.org/2001/XMLSchema`.

An attribute information item is a *DTD-determined ID* if and only if it has a `[type definition]` property whose value is equal to `ID`.

An *externally-determined ID* is a string, representing an element identifier, whose value is determined by the application through mechanisms outside the scope of this specification.

A shorthand pointer provides, for resources with XML-based media types, a rough analog of HTML fragment identifier behavior. However, if `ID` typing information is not available because no DTD, schema, or application-specific information is available, the pointer will not identify any element. There are several ways to make element identification more reliable. For example, the creator of a resource can use an internal DTD subset to indicate the presence of ID-typed attributes, and the creator of a pointer can, instead of a shorthand pointer, use a scheme-based pointer or provide one or more schemes that address the desired element in other ways.

The above definitions are *not* affected by whether or not the value which identified an element information item is unique within the document, because neither [XML] or [XMLSchema] require this for the assignment of type ID.

3.3. Scheme-Based Pointer

A scheme-based pointer consists of one or more pointer parts, optionally separated by white space (S). Each part has a scheme name and contains, within parentheses, data (EscapedData) conforming to the named scheme. If the scheme data contains parentheses, they must be either balanced or escaped.

When multiple pointer parts are provided, an XPointer processor must evaluate them in left-to-right order. If the XPointer processor does not support the scheme used in a pointer part, it skips that pointer part. If a pointer part does not identify any subresources, evaluation continues and the next pointer part, if any, is evaluated. The result of the first pointer part whose evaluation identifies one or more subresources is reported by the XPointer processor as the result of the pointer as a whole, and evaluation stops. If no pointer part identifies subresources, it is an error.

In the following example, if the 'xpointer' pointer part is not understood or fails to identify any subresources, the 'element' pointer part is evaluated. If the 'xpointer' pointer part identifies subresources, the 'element' pointer part is not evaluated.

```
#xpointer(id('boy-blue')/horn[1])element(boy-blue/3)
```

A scheme name consists syntactically of an optional Prefix and a LocalPart, as defined in [XML-Names]. Abstractly, scheme names are a tuple consisting of the LocalPart and the namespace name corresponding to that Prefix in the namespace binding context. If the namespace binding context contains no corresponding prefix, or if the (namespace name, LocalPart) pair does not correspond to a scheme name supported by the XPointer processor, the pointer part is skipped.

This specification reserves all unqualified scheme names for definition in additional XPointer schemes defined in W3C Recommendations. The use of QNames as scheme names provides a general framework for extensibility by other XML-based media types wishing to use this framework in defining their own fragment identifier languages. The definition of any scheme to be used in conjunction with the XPointer framework must specify a name for the scheme, consisting of a (namespace name, LocalPart) pair.

3.4. Namespace Binding Context

Scheme specifications may define ways to bind XML namespaces [XML-Names] prefixes to namespace names for the purpose of interpreting the prefixes of scheme names, or element names, attribute names and other QNames appearing in pointer parts. These bindings contribute to a namespace binding context that applies to all pointer parts to the right of the pointer part making the binding, unless exceptions are

explicitly made by the schemes in question. The documentation for any namespace-binding scheme must specify whether its bindings remain in effect for later pointer parts. The documentation for every scheme must specify whether it uses the namespace binding context.

In the following example, the 'xmlns' scheme (see [XPtrXmlns]) is used to add a (prefix/namespace name) binding to the namespace binding context. The XPointer processor uses this information to ascertain whether `img:rect` denotes the name of a scheme that it supports.

```
#xmlns(img=http://example.org/image)img:rect(10,10,50,50)
```

The initial namespace binding context prior to evaluation of the first pointer part consists of a single entry: the `xml` prefix bound to the namespace name `http://www.w3.org/XML/1998/namespace`. The namespace binding context is subject to the following constraints; attempts to violate these constraints will have no effect on the namespace binding context:

- The `xml` prefix is bound to the namespace name `http://www.w3.org/XML/1998/namespace`. It must not be bound to any other namespace name.
- The namespace name `http://www.w3.org/XML/1998/namespace` is bound to the prefix `xml`. It must not be bound to any other prefix.
- The `xmlns` prefix must not be bound to a namespace name.
- The namespace name `http://www.w3.org/2000/xmlns/` must not be bound to any prefix.

Prefixes beginning with the three-letter sequence x, m, l, in any case combination, are reserved. Users should not use them except as defined by XML and XML-related specifications.

4. Character Escaping

The set of characters for XPointers is [Unicode]. However, the XPointer language is designed to be used in the context of URI references [RFC 2396] and IRI references [IRI], which require encoding and escaping of certain characters. XPointers and IRI references containing XPointers also often appear in XML documents and external parsed entities, which impose some escaping requirements of their own when the encoding limits the repertoire that can be used directly. Other contexts might require additional escaping to be applied to XPointers. Also, because some characters are significant to XPointer processing, escaping is needed to use these characters in their ordinary sense.

4.1. Escaping Contexts

The following contexts require various types of escaping to be applied to XPointers:

A. Escaping of XPointer-significant characters

As described in § 3.1 – Syntax on page 3, unbalanced parentheses and occurrences of the circumflex must be escaped.

B. Escaping and Encoding of reserved IRI characters

An *Internationalized Resource Identifier*, or IRI is a protocol element that extends the syntax of URIs to a much wider repertoire of Unicode characters [Unicode]. IRI references allow a superset of the characters of fully escaped URI references, but must have normal occurrences of the percent sign (%) escaped because it is the character used for escaping in URIs and IRIs.

Thus, when a pointer is inserted into an IRI reference, any occurrences of percent signs (%) must be escaped. Other characters may be escaped as well, though it is not recommended. Characters are escaped as follows:

1. Each character to be escaped is converted to UTF-8 [RFC 2279] as one or more bytes.
2. The resulting bytes are escaped with the URI escaping mechanism (that is, converted to %HH, where HH is the hexadecimal notation of the byte value).
3. The original character is replaced by the resulting character sequence.

For example % becomes `%25`.

C. Escaping and Encoding of reserved URI characters

IRI references can be converted to URI references for consumption by URI resolvers. The disallowed characters in URI references include all non-ASCII characters, plus the excluded characters listed in Section 2.4 of [RFC 2396], except for the number sign (#) and percent sign (%) and the square bracket characters re-allowed in [RFC 2732]. Disallowed characters are escaped as follows:

1. Each disallowed character is converted to UTF-8 [RFC 2279] as one or more bytes.
2. The resulting bytes are escaped with the URI escaping mechanism (that is, converted to %HH, where HH is the hexadecimal notation of the byte value).
3. The original character is replaced by the resulting character sequence.

D. XML escaping

If a pointer appears in an XML document or external parsed entity, any characters not expressible in the encoding used must be escaped as character references, and any characters that are significant to XML processing at the point where they appear must be escaped by an appropriate mechanism such as character references or entity references. This escaping is reversed when the XML document or entity is parsed. It is not recommended that URI references (rather than the more general IRI references) be placed in XML documents. If for some reason this proves unavoidable, the same escaping mechanism applies.

Since the XPointer processor will reverse only the escaping of XPointer-significant characters (A), the application must reverse any other encodings or escapings (such as B, C, or D) that the pointer was subject to. If the result passed to the XPointer processor does not conform to the syntactic rules for XPointers in this specification, it is an error.

4.2. Examples of Escaping

The following table shows the escaping in various contexts of an XPointer containing an unbalanced parenthesis, double quotation marks, and spaces. These examples use the 'xpointer' scheme (see [XPtrX-Pointer]), since it allows string literals in its scheme data.

Context	Notation
Initial Scheme Data	The `xpointer` scheme data as it was initially created: `string-range(//P,"my favorite smiley :-)")`

Context	Notation
A. XPointer	With the unbalanced parenthesis in the scheme data escaped, as required by this specification: `xpointer(string-range(//P,"my favorite smiley :-^)"))`
B. Pointer in IRI reference	Same as A (no percent sign found that needs escaping): `#xpointer(string-range(//P,"my favorite smiley :-^)"))`
C. IRI reference converted to URI reference	With occurrences of the double quotatation marks (`%22`), spaces (`%20`), and circumflexes (`%5E`) escaped: `#xpointer(string-range(//P,%22my%20favorite%20smiley%20:-%5E)%22))`
D. IRI reference in XML document	Double quotation marks escaped using XML's predefined `&quot;` entity reference (assuming that the pointer appears in an IRI reference in a double-quoted attribute value): `#xpointer(string-range(//P,&quot;my favorite smiley :-^)&quot;))`

The following table shows the escaping of an XPointer containing accented characters in various contexts. The XML document is assumed to be encoded in `US-ASCII`, which does not allow the letter “é” to appear directly.

Context	Notation
Initial Scheme Data	The `xpointer` scheme data as it was initially created: `id('résumé')`
A. XPointer	The XPointer (no circumflexes or unbalanced parentheses in scheme data that need escaping): `xpointer(id('résumé'))`
B. Pointer in IRI reference	Same as A (no percent sign found that needs escaping): `#xpointer(id('résumé'))`
C. IRI reference converted to URI reference	With occurrences of the letter “é” (`%C3%A9`) escaped: `#xpointer(id('r%C3%A9sum%C3%A9'))`
D. IRI reference in XML document	Represented in the `US-ASCII` encoding; accented letters are escaped with XML character references: `#xpointer(id('r&#xE9;sum&#xE9;'))`

Appendix A. References

A.1. Normative References

Infoset

John Cowan and Richard Tobin, editors. *XML Information Set*. World Wide Web Consortium, 2001.

RFC 2119

Scott Bradner, *RFC 2119: Key words for use in RFCs to Indicate Requirement Levels*. Internet Engineering Task Force, 1997.

RFC 2396

Tim Berners-Lee, Roy Fielding, and Larry Masinter, *RFC 2396: Uniform Resource Identifiers*. Internet Engineering Task Force, 1995.

RFC 2732

Robert Hinden, Brian Carpenter, and Larry Masinter, *RFC 2732: Format for Literal IPv6 Addresses in URL's*. Internet Engineering Task Force, 1999.

RFC 2279

François Yergeau *RFC 2279: UTF-8, a transformation format of ISO 10646*. Internet Engineering Task Force, 1998.

RFC 3023

MURATA Makoto, Simon St.Laurent, and Dan Kohn, *RFC 3023: XML Media Types*. Internet Engineering Task Force, 2001.

XML

Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, and Eve Maler, editors. *Extensible Markup Language (XML) 1.0 (Second Edition).* World Wide Web Consortium, 2000.

XML-Names

Tim Bray, Dave Hollander, and Andrew Layman, editors. *Namespaces in XML*. World Wide Web Consortium, 1999.

XMLSchema

Henry Thompson et al., editors. *XML Schema Part 1.* World Wide Web Consortium, 2001.

Unicode

The Unicode Consortium *The Unicode Standard*.

A.2. Non-Normative References

HTML

Dave Raggett, Arnaud Le Hors, and Ian Jacobs. *HTML 4.01 Specification*. World Wide Web Consortium, 1999.

IRI

Martin J. Dürst and Michel Suignard *Internationalized Resource Identifiers (IRI)* Internet draft draft-duerst-iri-01, Internet Engineering Task Force, 2002. *Work in progress*.

RDF

Dave Beckett, editor. *RDF/XML Syntax Specification*. World Wide Web Consortium, 2001.

SOAP12

Nilo Mitra et al., editors. *SOAP Version 1.2* Parts 0, 1, and 2. World Wide Web Consortium, 2001. *Work in progress.*

XInclude

Jonathan Marsh and David Orchard, editors. *XML Inclusions (XInclude) Version 1.0*. *Work in progress*. World Wide Web Consortium, 2001.

XLink

Steve DeRose, Eve Maler, and David Orchard, editors. *XML Linking Language (XLink)*. World Wide Web Consortium, 2001.

XPtrXmlns

Paul Grosso, Eve Maler, Jonathan Marsh, and Norman Walsh, editors. *XPointer xmlns() Scheme*. World Wide Web Consortium, 2003.

XPtrXPointer

Steven DeRose, Eve Maler, and Ron Daniel Jr., editors. *XPointer xpointer() Scheme*. World Wide Web Consortium, 2002. *Work in progress.*