XPath Requirements Version 2.0

W3C Working Draft 14 February 2001

This version:
http://www.w3.org/TR/2001/WD-xpath20req-20010214
(available in XML or HTML)
Latest version:
http://www.w3.org/TR/xpath20req
Editors:
Steve Muench (Oracle) <Steve.Muench@oracle.com>
Mark Scardina (Oracle) <Mark.Scardina@oracle.com>
Mary Fernandez (AT&T) <mff@research.att.com>

Abstract

This document describes the requirements for the XPath 2.0 specification.

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 the first public XPath 2.0 Requirements working draft.

This is a W3C Working Draft for review by W3C Members and other interested parties. It is a draft document and may be updated, replaced or made obsolete by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress". This is work in progress and does not imply endorsement by the W3C membership.

This document has been produced jointly as part of the W3C Style Activity and the W3C XML Activity, following the procedures set out for the W3C Process. The document has been written by the XSL Working Group (W3C members only) and the XML Query Working Group (W3C members only). The goals of the XSL Working Group are discussed in the XSL Working Group charter, and the goals of the XML Query Working Group are discussed in the XML Query Working Group Charter. This Working Draft represents the current thinking of the XSL Working Group and XML Query Working Group. The groups have consensus except in a few specific areas as noted below and the details of these areas are still under active discussion. Nonetheless, we are publishing this draft to encourage early public feedback to the XPath 2.0 requirements process.

Comments on this document should be sent to the W3C mailing list xsl-editors@w3.org (archived at http://lists.w3.org/Archives/Public/xsl-editors/). A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR/.

Table of contents

---

1 Goals

XPath 2.0 has the following goals:

- Simplify manipulation of XML Schema-typed content
- Simplify manipulation of string content
- Support related XML standards
- Improve ease of use
- Improve interoperability
- Improve i18n support
- Maintain backward compatibility
- Enable improved processor efficiency

2 Requirements

1 Must Support the XML "Family" of Standards

As part of the evolving family of XML standards, XPath 2.0 MUST support the W3C XML architecture by integrating well with other standards in the family.

1.1 Must Express Data model in Terms of the Infoset

XPath 2.0 data model MUST be expressed in terms of the XML Infoset, including Post Schema Validation Infoset contributions. Ideally, XSLT, XPath, and XML Query should share a common data model.

1.2 Must Provide Common Core Syntax and Semantics for XSLT 2.0 and XML Query 1.0

XSLT 2.0 contains an expression language which is "XPath 2.0 plus XSLT 2.0 extensions". XML Query 1.0 contains an expression language which we believe should be "XPath 2.0 plus XML Query 1.0 extensions." The scope of XPath 2.0 must be the set of common functionality between the expression language of XSLT 2.0 and the expression language of XML Query 1.0. This will guarantee for Web developers and users that any common functionality are implemented with identical syntax and semantics.

> **Ed. Note:** The XSL Working group has consensus on the above requirement but the XML Query Working Group does not. Specifically, the XML Query Working Group is discussing what functionality constitutes a common core.

The Working Groups therefore particularly wish to receive feedback on this requirement.

1.3 Must Support Explicit "For Any" or "For All" Comparison and Equality Semantics

It MUST be possible for boolean expressions involving node-sets to explicitly use "for any" or "for all" semantics, otherwise known as explicit existential quantification and explicit universal quantification, respectively. Any *implicit* quantification should be explained in terms of these explicit semantics.

Use Case

The following examples illustrate limitations of XPath 1.0 that can be addressed by the addition of existential and universal quantifiers to XPath 2.0:

1. In XPath 1.0, the expression `5 > $ns` evaluates to true if `5` is greater than the `number()` value of at least one node in the `$ns` node set. There is no way to test whether `5` is greater than the `number()` values of all nodes in `$ns`.
2. In XPath 1.0, the expression `$ns1 > $ns2` evaluates to true if the `number()` value of at least one node in `$ns1` is greater than the `number()` value of at least one node in the `$ns2` node set. There is no way to test whether the `number()` values of all of the nodes in `$ns1` are greater than all the `number()` values of all nodes in `$ns2`.
3. In XPath 1.0, the expression `5 = $ns` evaluates to true if `5` equals the `number()` value of at least one node in the `$ns` node set. There is no way to test whether `5` equals the `number()` value of all nodes in `$ns`.
4. In XPath 1.0, the expression `$ns1 = $ns2` evaluates to true if the `number()` value of at least one node in `$ns1` equals the `number()` value of at least one node in the `$ns2` node set. There is no way to test whether the `number()` values of all of the nodes in `$ns1` are greater than all the `number()` values of all nodes in `$ns2`.
5. In XPath 1.0, the expression `0 + LineItem/@UnitPrice > 40` is equivalent to `0 + number(OrderDetail/@UnitPrice) > 40` which will be true if the *first* `LineItem` element's `UnitPrice` attribute is greater than 40. However, the expression `LineItem/@UnitPrice > 40` is true if *any* of the `LineItem` elements' `UnitPrice` attributes is greater than 40. This difference in behavior could be resolved through explicit quantification.

1.4 Must Extend Set of Aggregation Functions

XPath 1.0 currently supports `sum()` and `count()`. XSLT users have frequently requested the addition of `min()` and `max()`. XPath 2.0 MUST address this by extending its basic set of aggegration functions by adopting some or all of the ones defined for XML Query.

1.5 Should Maintain Backwards Compatibility with XPath 1.0

Any valid XPath 1.0 expression SHOULD also be a valid XPath 2.0 expression, and have the same semantics when operating in the absence of XML Schema type information. If not possible to achieve due to other requirements, XPath 2.0 should minimize the number of changes to the XPath 2.0 syntax to maximize backward compatibility with XPath 1.0.

1.6 Should Provide Intersection and Difference Functions

XPath 1.0 supports the union of two node sets. Set functionality in XPath 2.0 SHOULD be expanded to include intersection and difference functions, and should be compatible with equivalent functions supported by XML Query.

Use Case

1. Given an XML document like:

```
<hr>
  <teams>
    <team id="t1">
      <members ref="e101 e103"/>
    </team>
    <team id="t2">
      <members ref="e102"/>
    </team>
  </teams>
  <departments>
    <department>
      <employee id="e101" name="Steve"    teams="t1"/>
      <employee id="e102" name="Jonathan" teams="t2"/>
      <employee id="e103" name="James"    teams="t1"/>
    </department>
  </departments>
</hr>
```

   Format the list of employees that work in the same department as the `employee` with `id` = `e101` and that are on the same teams as that employee. To accomplish this:

   - Select the list of all employees in the employee's department:

     ```
     id('e101')/parent::department/employee
     ```

   - Intersect that set with the list of all employees who are members of the current employee's teams:

     ```
     id('e101')/id(id(@teams)/members/@ref)
     ```

   - Difference that set with the list containing the current employee so they don't appear in their own list of team members:

     ```
     id('e101')
     ```

   The result should be the employee "James".

2. Given an XML document full of book information, a book with multiple authors appears like this:

```
<books>
  <book>
    <title>Exciting New Book</title>
    <author>Kay</author>
```

```
    <autnor>Muencn</autnor>
  </book>
  <book>
    <title>Building Oracle XML Applications</title>
    <author>Muench</author>
  </book>
  <book>
    <title>XSLT Programmer's Reference</title>
    <author>Kay</author>
  </book>
</books>
```

Assuming an XSLT key is defined as follows:

```
<xsl:key name="auth" match="book" use="author"/>
```

then, to find the list of books that have both Muench and Kay among their authors, you need to select the intersection of `key('auth','Muench')` and `key('auth','Kay')`.

To find books authored by Muench but *not* by Kay, you need to select the difference of `key('auth','Muench')` and `key('auth','Kay')`.

1.7 Should Support Unary Plus Operator

XML Schema allows decimals to have a leading plus. To align better with this, XPath 2.0 SHOULD allow this by providing a unary plus operator.

> **Ed. Note:** There's a potential though rather remote problem if there are nodes in the source document containing a number with a leading plus sign. Currently the defined behavior is to convert these values to `NaN`, and this will change. A stylesheet that is looking for phone-numbers beginning with "+" might rely on this.

2 Must Improve Ease of Use

Users of XPath 1.0 have requested enhancements to simplify expression of common XPath use cases. XPath 2.0 MUST address these frequently requested enhancements.

2.1 Must Loosen Restrictions on Location Steps

To better align with XPointer and to simplify the use of XPath expressions in which multiple alternatives are allowed for a given location step, XPath 2.0 MUST relax current restrictions for what can appear after a '/' in an XPath expression. Neither unions nor node-set functions are allowed to appear after a '/' in XPath 1.0.

Use Case

1. An XPointer's expression can use node-set functions on the right of the '/' in a path:

   ```
   id("chap1")/range-to(id("chap2"))
   ```

2. Allowing unions in a location step would allow expressions like:

   ```
   /foo/(xxx|yyy)/li/(p|eq)
   ```

instead of the more cumbersome:

```
/foo/xxx/li/p  |
/foo/xxx/li/eg |
/foo/yyy/li/p  |
/foo/yyy/li/eg
```

3. Allowing node-set functions to appear in a location step would allow expressions like:

```
document("otherdoc.xml")/id("foo")/a/b
```

and

```
document("otherdoc.xml")/key("x","foo")/a/b
```

2.2 Must Provide a Conditional Expression

Many users have requested the ability to return a conditional value based on a boolean expression. XPath 2.0 MUST provide a conditional expression which takes three expressions:

1. *expression1 (boolean)*
2. *expression2*
3. *expression3*

and evaluates to *expression2* if *expression1* is `true` and to *expression3* if `expression1` is `false`. Only *expression1* and the expression to be returned must be evaluated.

Use Case

XSLT Use Cases

1. In a template, construct a `<table>` element with a `border` attribute whose value is provided by an *attribute value template* and is conditional depending on the value of the `frame` attribute of the current node. If `@frame = 'none'`, then the value of `border` should be `0`, otherwise the value of `border` should be `1`.
2. Assign an XSLT variable a conditional value based on whether the current node has any `<test>` element children. If it does, assign the value "`is`" to the variable, otherwise assign the value "`is not`".
3. Given a source XML document like this:

```
<Emps>
  <Emp>
     <LastName>Smith</LastName>
  </Emp>
  <Emp>
    <CommonName>Scott</CommonName>
  </Emp>
</Emps>
```

provide an `<xsl:sort>` select expression to sort the list of employees on the value of their `<LastName>` if it's present, otherwise on the value of their

value of their <LastName> if it's present, otherwise on the value of their <CommonName> element.

2.3 Must Define Consistent Implicit Semantics for Collection-Valued Subexpressions

In XPath 1.0, the use of a collection-valued subexpression can introduce an implicit existential quantification or choose-first-member operation into the containing expression's semantics. XPath 2.0 MUST define a consistent implicit semantics for expressions that have collection-valued subexpressions. This may require that XPath 2.0 expressions use explicit quantification or indexing expressions to achieve the same implicit semantics provided in XPath 1.0.

Use Case

1. In XPath 1.0, the expression `a[b=5]` returns <a> elements that have ANY <b> element child with value 5, where as the expression `a[b+1=6]` returns <a> elements whose FIRST <b> element child has value 5. These results should be consistent for XPath 2.0.
2. Assume for this use case that the typed-value of an element or attribute is referenced using the syntax `typed-value(elt)` and `typed-value(@attr)`, respectively. In XPath 2.0, when working with the typed value of an element or attribute with `boolean` type, the expression `a[typed-value(b) = false()]` should return <a> elements having at least one <b> element child with `boolean` value `false`. Similarly, the expression `a[typed-value(@b) = false()]` should return <a> elements having a boolean attribute `b` with value `false`.

2.4 Should Support Additional String Functions

Users of XSLT 1.0 have requested additional string manipulation functions. Common requests include string padding (with spaces, dashes, or other characters), string replacement, and converting strings to upper and lower case. XPath 2.0 SHOULD provide additional string functions. As with any string functions, internationalization issues need to be addressed to insure that the functionality is as broadly useful as possible.

2.4.1 Should Simplify String Replacement

XPath 1.0 provides the `translate()` function that allows each single character in a source string to be translated to another single character in the result string, or to be removed. Users of XPath 1.0 frequently want to replace *sequences* of consecutive characters with other sequences of characters, which `translate()` does not support. XPath 2.0 SHOULD support a more flexible string replacement function.

Use Case

Replace each occurrence of "`foo`" in a string with "`foobar`".

2.4.2 Should Simplify String Padding

Often string values need to be padded on the left or right to make the value occupy a fixed length. XPath 2.0 SHOULD support a string padding function that permits any character as a padding character.

Use Case

1. Pad the value of a string on the right to ensure a length of 10 characters using spaces
2. Pad the value of a string on the left to 10 characters using asterisks
3. Output a string containing as many dashes as the length of a section title

2.4.3 Should Simplify String Case Conversions

XPath 2.0 SHOULD provide the ability to convert the case of text to upper or lower case for presentation and/or comparison.

2.5 Should Support Aggregation Functions Over Collection-Valued Expressions

Users of XPath 1.0 frequently request the ability to apply an aggregate function, like `sum()`, to the values of expressions applied to a node set. XPath 2.0 SHOULD support aggregation functions over expressions applied to node sets.

Use Case

1. Given a document like

```
<sect title="Plan">
   <sect title="Overview">
      <sect title="Competitors">
        <sect title="SuperSoft"/>
        <sect title="BarnWare"/>
      </sect>
   </sect>
   <sect title="Details">
      <sect title="Summary"/>
   </sect>
</sect>
```

   Calculate the maximum depth of the nesting of `<sect>` elements. The nodeset is identified by the expression `//sect` and the expression to maximize would be `count(ancestor-or-self::sect)`.

2. Given the same document above, calculate the average string length of section titles. The nodeset is identified by `//section/@title` and the expression to sum would be `string-length(.)`, divided by the `count(//section/@title)`
3. Given an XML document containing a purchase order and its line `<item>` elements, calculate the total amount of the purchase order by summing the price times the quantity of each item. The nodeset is identified by `item`, and the expression to sum would be `price * quantity`.

3 Must Support String Matching Using Regular Expressions

Regular expressions provide a powerful way to specify string pattern matching and now play an important role in XML Schema as the mechanism by which pattern facets are specified. XPath 2.0 MUST support regular expressions for matching against strings using the regular expression notation established in XML Schema: Datatypes.

Use Case

1. Given XML like:

```
<Data>
  <EmpRow>
     :
  </EmpRow>
  <DeptRow>
     <Deptno>10</Deptno>
     <Employees>
        <EmpRow>
          <Empno>1000</Empno>
          <Ename>1000</Ename>
        </Emprow>
     </Employees>
  </DeptRow>
</Data>
```

   match elements that "end with `Row` or `row`"

```
*[local-name() =~ ".*[rR]ow"]
```

2. Match phone numbers like 123-456-7890

```
field[ . =~ "\d\d\d-\d\d\d\-\d\d\d\d ]
```

3. Match any element name with the string "Addr" contained within the name (case-insensitively)

```
*[local-name() =~ ".*[Aa][Dd][Dd][Rr].*"]
```

4. Match a string from a dynamic regular expression

```
Department[Code =~ concat("[0-9][0-9]",$v,".*")]
```

4 Must Add Support for XML Schema Primitive Datatypes

XML Schema: Datatypes defines a set of primitive datatypes. In addition to the types supported by the XPath 1.0 data model, `string`, `number`, `boolean`, and `node-set`, the XPath 2.0 data model MUST support XML Schema primitive types. XPath 2.0 will extend the XPath data model to accommodate working with elements and attribute values having an XML Schema primitive type.

4.1 Must Define the Operator Matrix and Conversions

XPath 2.0 MUST support the operators and type-coercion rules defined by the joint XSLT/Schema/Query task force on operators.

Use Case

- **General**

1. Use and create the type information of any datatype
2. Do not require the user to respecify the datatype (i.e. number and boolean)
3. Maintain the facet constraints of primitive datatypes while manipulating the value. For example, take a 14.4 number plus a 14.4 number and return a 14.4 number.

- **For DateTime**

1. Compare dates of documents and return a boolean
2. Add a time or date interval and return the same type
3. Add a scalar value to a time or date and return the same type
4. Calculate an end time from a start time+duration. For example, given:

```
<appointment time="20001224-16:00:00"
duration="1:00"/>
```

Produce the end-time:

```
End time: <xsl:value-of select="@time + @duration"/>
```

5. Detect intersections between datetime durations

```
<appointment time="20001224-16:00:00"
             duration="2:00" reminder="0:15"/>
<appointment time="20001224-17:00:00"
             duration="1:00" reminder="0:15"/>
```

- **For Boolean**

Perform pattern matching on "yes" and "no"

- **For QName**

1. Build a Namespace element from a QName
2. Compare a Namespace URI from a QName with a string

4.2 Must Allow Scientific Notation for Numbers

XML Schema specifies a lexical representation for doubles and floats that includes scientific notation as well as `INF`, `-INF` or `NaN`. XPath 2.0 MUST support the lexical representations of floats and doubles supported by XML Schema.

4.3 Must Define Appropriate Cast and Constructor Functions

XPath 2.0 MUST define an appropriate set of functions to allow users to cast and construct instances of XML Schema primitive types. At a minimum, this set MUST include cast and constructor functions for URI and date/time types.

Use Case

Be able to compare an element or attribute with an XML Schema date or time type with a constant date or time value.

4.4 Should Add List Data Type to the Type System of the Expression Langauge

XML Schema allows the definition of simple types derived by list, including lists of unions of non-list simple types. XPath 2.0 SHOULD support an ordered list of simple-typed values.

Use Case

Create an XSLT transformation that converts an element structure like this:

```
<xdr:attribute name="a1"
                    dt:type="enumeration"
                    dt:values="a1 a2 a3"/>
```

into an alternative structure like this:

```
<xsd:attribute name="a1">
  <xsd:simpleType>
    <xsd:restriction base="NMTOKEN">
      <xsd:enumeration value="a1"/>
      <xsd:enumeration value="a2"/>
      <xsd:enumeration value="a3"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
```

This requires the ability in XPath to select the list of NMTOKENS values of the `dt:values` attribute for processing.

4.5 Must Support Accessing Simple-Typed Value of Elements and Attributes

The XPath 1.0 type system supports the `number`, `string`, `boolean`, and `node-set` types. XML Schema: Datatypes introduces many new types. XPath 1.0 supports conversion of the simple-typed values of elements and attributes to strings. In addition to this functionality, XPath 2.0 MUST support access to the native, simple-typed value of an element or attribute.

4.6 Must Define Behavior of Operators for Null Arguments

Since the typed value of an element can be `null`, XPath 2.0 MUST define how the behavior of operations applies to null values.

Use Case

Assuming the typed value of an element is addressed using a syntax like `typed-value (elementname)`, XPath needs to specify what `1.20 * typed-value(salary)` evaluates to when the typed value of the `<salary>` is `null`.

5 Should Add Support for XML Schema: Structures

XML Schema: Structures enables users to define structured types and associate them to elements in a schema. XPath 2.0 SHOULD provide support for the common operations needed for navigation and selection of typed elements.

5.1 Should Select Elements/Attributes Based on an Explicit XML Schema Type

XML Schema : Structures provides the ability to define the type of an element or attribute. XPath 2.0 SHOULD be able to test whether an element or attribute is an instance of a given type.

Use Case

1. Select all elements that are instances of the complex type `Address`.
2. Select all attributes that are instances of the simple-type `xsd:integer`.
3. Select elements of type `Address` indicated in the instance with `xsi:type="Address"`

5.2 Should Select Elements/Attributes Based on XML Schema Type Hierarchy

XML Schema : Structures provides the ability to define a hierarchy of types by derivation. XPath 2.0 SHOULD be able to selects elements or attributes that are instances of a type, also matching any types derived from it by restriction or extension.

> **Ed. Note:** This introduces some interesting precedence decisions about how matches for an explicit type would need to take precedence over matches for compatible types. In other words, given a template matching elements with types compatible to `Address` and another template explicitly matching type `USAddress`, one would expect the explicit type match to take precedence. Also consider the case where two templates exist, one that matches the `Address` type and one that matches the `USAddress` type. If an element of type RuralRouteAddress (further derived from USAddress) is considered for matching, one would expect the template for the "closest inherited ancestor", that is `USAddress`, to take precendence over a template matching the type of a "more distant inherited ancestor" like `Address`.

Use Case

Derivation by Restriction

The XML Schema Primer defines the `Items` type and the `ConfirmedItems` type which is derived by restriction from `Items`. Select purchase orders whose `<items>` member element is an instance of the `ConfirmedItems` type, which restricts the minimum number of `<item>` elements that can appear to be at least one.

Derivation by Extension

The XML Schema Primer defines the `Address` type and the `UKAddress` and `USAddress` types which is derived by extension from `Address`, adding additional element content in the subtypes.

1. Select all elements that are instances of complex type `Address`, matching elements of type `USAddress` and elements of type `UKAddress` as well.
2. Select all elements that are instances of complex type `Address`, without selecting elements of types derived from `Address`.
3. Select all attributes that are instances of the simple-type `xsd:integer`, matching attribute of type `SKU-Number` which derives from `xsd:integer`.

5.3 Should Select Elements Based on XML Schema Substitution Groups

XML Schema: Datatypes provides the ability to include two or more element names in a substitution group. XPath 2.0 SHOULD be able to test whether an element is a member of an XML Schema substitution group.

Use Case

The XML Schema Primer defines the global `<ipo:comment>` element as the head of a substitution group, whose substitutible member elements include `<ipo:customerComment>` and `<ipo:shipComment>`. Select the list of all elements that are `<ipo:comment>` elements, including any elements that are substitutible for `<ipo:comment>` through this substitution group.

5.4 Should Support Lookups Based on Schema Unique Constraints and Keys

XML Schema: Datatypes supports named, multi-part keys. XPath 2.0 SHOULD support a mechanism for looking up the element to which a Schema key refers. Similar mechanisms already exist, such as the `id()` function in XPath 1.0 and the `key()` function in XSLT.

3 References

XML Schema: Structures, http://www.w3.org/TR/xmlschema-1/
XML Infoset, http://www.w3.org/TR/xml-infoset
XML Schema: Datatypes, http://www.w3.org/TR/xmlschema-2/
XPointer, http://www.w3.org/TR/xptr