Section: *Axes*

4.6.4.1 Axes

Changes in 4.0 ↓ ↑

1. Four new axes have been defined: `preceding-or-self`, `preceding-sibling-or-self`, `following-or-self`, and `following-sibling-or-self`. *[Issue 1519]*

ForwardAxis	::=	("attribute" | "child" | "descendant" | "descendant-or-self" | "following" | "following-or-self" | "following-sibling" | "following-sibling-or-self" | "namespace" | "self") "::"
ReverseAxis	::=	("ancestor" | "ancestor-or-self" | "parent" | "preceding" | "preceding-or-self" | "preceding-sibling-or-self") "::"
BidirectionalAxis	::=	“sibling::”

XPath defines a set of **axes** for traversing documents, but a **host language** may define a subset of these axes. The following axes are defined:

- The `child` axis contains the children of the context node, which are the nodes returned by the Section 4.3 children Accessor*DM*.
  Note:
  Only document nodes and element nodes have children. If the context node is any other kind of node, or if the context node is an empty document or element node, then the child axis is an empty sequence. The children of a document node or element node may be element, processing instruction, comment, or text nodes. Attribute, namespace, and document nodes can never appear as children.

- The `descendant` axis is defined as the transitive closure of the child axis; it contains the descendants of the context node (the children, the children of the children, and so on).
  More formally, `$node/descendant::node()` delivers the result of `fn:transitive-closure($node, fn { child::node() })`.

- The descendant-or-self axis contains the context node and the descendants of the context node.
  More formally, $node/descendant-or-self::node() delivers the result of $node/(. | descendant::node()).
- The parent axis contains the sequence returned by the Section 4.11 parent Accessor^DM, which returns the parent of the context node, or an empty sequence if the context node has no parent.
  Note:
  An attribute node may have an element node as its parent, even though the attribute node is not a child of the element node.
- The ancestor axis is defined as the transitive closure of the parent axis; it contains the ancestors of the context node (the parent, the parent of the parent, and so on).
  More formally, $node/ancestor::node() delivers the result of fn:transitive-closure($node, fn { parent::node() }).
  Note:
  The ancestor axis includes the root node of the tree in which the context node is found, unless the context node is the root node.
- The ancestor-or-self axis contains the context node and the ancestors of the context node; thus, the ancestor-or-self axis will always include the root node.
  More formally, $node/ancestor-or-self::node() delivers the result of $node/(. | ancestor::node()).
- The following-sibling axis contains the context node’s following siblings, that is, those children of the context node’s parent that occur after the context node in document order. If the context node is an attribute or namespace node, the following-sibling axis is empty.
  More formally, $node/following-sibling::node() delivers the result of fn:siblings($node)[. >> $node]).
- The following-sibling-or-self axis contains the context node, together with the contents of the following-sibling axis.
  More formally, $node/following-sibling-or-self::node() delivers the result of fn:siblings($node)[not(. << $node)]
- The preceding-sibling axis contains the context node’s preceding siblings, that is, those children of the context node’s parent that occur before the context node in document order. If the context node is an attribute or namespace node, the preceding-sibling axis is empty.
  More formally, $node/preceding-sibling::node() delivers the result of fn:siblings($node)[. << $node].
- The preceding-sibling-or-self axis contains the context node, together with the contents of the preceding-sibling axis.

More formally, $node/preceding-sibling-or-self::node() delivers the result of fn:siblings($node)[not(. >> $node).

- The sibling axis contains the context node's preceding siblings and its following siblings, that is, those children of the context node's parent that occur before the context node and after the context node – in document order.
  More formally, $node/sibling::node() delivers the result of fn:siblings($node)[not(. is $node)]
- The following axis contains all nodes that are descendants of the root of the tree in which the context node is found, are not descendants of the context node, and occur after the context node in document order.
  More formally, $node/following::node() delivers the result of $node/ancestor-or-self::node()/following-sibling::node()/descendant-or-self::node()
- The following-or-self axis contains the context node, together with the contents of the following axis.
  More formally, $node/following-or-self::node() delivers the result of $node/(. | following::node()).
- The preceding axis contains all nodes that are descendants of the root of the tree in which the context node is found, are not ancestors of the context node, and occur before the context node in document order.
  More formally, $node/preceding::node() delivers the result of $node/ancestor-or-self::node()/preceding-sibling::node()/descendant-or-self::node().
- The preceding-or-self axis contains the context node, together with the contents of the preceding axis.
  More formally, $node/preceding-or-self::node() delivers the result of $node/(. | preceding::node()).
- The attribute axis contains the attributes of the context node, which are the nodes returned by the Section 4.1 attributes AccessorDM ; the axis will be empty unless the context node is an element.
- The self axis contains just the context node itself.
  The self axis is primarily useful when testing whether the context node satisfies particular conditions, for example if ($x[self::chapter]).
  More formally, $node/self::node() delivers the result of $node.
- The namespace axis contains the namespace nodes of the context node, which are the nodes returned by the Section 4.7 namespace-nodes AccessorDM; this axis is empty unless the context node is an element node. The namespace axis is deprecated as of XPath 2.0. If XPath 1.0 compatibility mode is true, the namespace axis must be supported. If XPath 1.0 compatibility mode is false, then support for the namespace axis is implementation-defined. An implementation that does not support the namespace axis when XPath 1.0 compatibility mode is false must raise a static error [err:XPST0010] if it is used. Applications needing information about the in-scope namespaces of an element should use the functions Section 10.2.7 fn:in-scope-prefixesFO, and Section 10.2.8 fn:namespace-uri-for-prefixFO.

Axes can be categorized as **forward axes**, **reverse axes** and **bidirectional axes**. An axis that only ever contains the context node or nodes that are after the context node in document order is a forward axis. An axis that only ever contains the context node or nodes that are before the context node in document order is a reverse axis.

A bidirectional axis bidi-axis can conceptually be defined as the union of two conceptual axes: left-bidi-axis and right-bidi-axis; left-bidi-axis being a reverse axis and right-bidi-axis being a forward axis. The left-bidi-axis consists of all nodes from bidi-axis that are not following the context node. The right-bidi-axis consists of all nodes from bidi-axis that are not preceding the context node.
The context position relative to a context node belonging to a bidi-axis is a non-zero integer: negative, if the node is contained in the left-bidi-axis, or positive, if the node is contained in the right-bidi-axis. By definition, fn:last-left() produces the negative value -left-bidi-axis::test/fn:last(). The context-size is defined as: bidi-axis::test/(fn:last() – fn:last-left())

The parent, ancestor, ancestor-or-self, preceding, preceding-or-self, preceding-sibling, and preceding-sibling-or-self axes are reverse axes; the sibling axis is a bidirectional axis; all other axes are forward axes.
The ancestor, descendant, following, preceding and self axes partition a document (ignoring attribute and namespace nodes): they do not overlap and together they contain all the nodes in the document.

[Definition: Every axis has a **principal node kind**. If an axis can contain elements, then the principal node kind is element; otherwise, it is the kind of nodes that the axis can contain.] Thus:

- For the attribute axis, the principal node kind is attribute.
- For the namespace axis, the principal node kind is namespace.
- For all other axes, the principal node kind is element.

Section: Predicates within Steps

4.6.5 Predicates within Steps

AxisStep	::=	(ReverseStep | ForwardStep | BidirectionalStep) Predicate*
Predicate	::=	"[" Expr "]"

A predicate within a AxisStep has similar syntax and semantics to a predicate within a filter expression. The only difference is in the way the context position is set for evaluation of the predicate.

Note:

The operator `[]` binds more tightly than `/`. This means that the expression `a/b[1]` is interpreted as `child::a/(child::b[1])`: it selects the first `b` child of every `a` element, in contrast to `(a/b)[1]` which selects the first `b` element that is a child of some `a` element.
A common mistake is to write `//a[1]` where `(//a)[1]` is intended. The first expression, `//a[1]`, selects every descendant `a` element that is the first child of its parent (it expands to `/descendant-or-self::node()/child::a[1]`), whereas `(//a)[1]` selects the `a` element in the document.

For the purpose of evaluating the context position within a predicate, the input sequence is considered to be sorted as follows: into document order if the predicate is in a (forward or bidirectional)-axis step, into reverse document order if the predicate is in a reverse-axis step, or in its original order if the predicate is not in a step.

More formally:

- For a step using a forwards or a bidirectional axis, such as `child::`*test*`[`*P*`]`, the result is the same as for the equivalent filter expression `(child::`*test*`)[`*P*`]` (note the parentheses). The same applies if there are multiple predicates, for example `child::`*test*`[`P_1`][`P_2`][`P_3`]` is equivalent to `(child::`*test*`)[`P_1`][`P_2`][`P_3`]`.
- For a step using a reverse axis, such as `ancestor::`*test*`[`*P*`]`, the result is the same as the expression `reverse(ancestor::`*test*`)[`*P*`] => reverse()`. The same applies if there are multiple predicates, for example `ancestor::`*test*`[`P_1`][`P_2`][`P_3`]` is equivalent to `reverse(ancestor::`*test*`)[`P_1`][`P_2`][`P_3`] => reverse()`.

Note:

The result of the expression `preceding-sibling::*` is in document order, but `preceding-sibling::*[1]` selects the last preceding sibling element, that is, the one that immediately precedes the context node.
Similarly, the expression `preceding-sibling::x[1,2,3]` selects the last three preceding siblings, returning them in document order. For example, given the input:

```
<doc><a/><b/><c/><d/><e/><f/></doc>
```

The result of `//e ! preceding-sibling::*[1,2,3]` is `<b/>, <c/>, <d/>`. The expression `//e ! preceding-sibling::*[3,2,1]` delivers exactly the same result.
The result of the expression

Here are some examples of axis steps that contain predicates:

- This example selects the second chapter element that is a child of the context node:
  child::chapter[2]
- This example selects all the descendants of the context node that are elements named "toy" and whose color attribute has the value "red":
  descendant::toy[attribute::color = "red"]
- This example selects all the employee children of the context node that have both a secretary child element and an assistant child element:
  child::employee[secretary][assistant]
- This example selects all siblings of the context node, that are a person element, whose gender attribute has the value "m":
  sibling::person [@gender eq "m"]
- This example selects the third following sibling of the context node, that is an element:
  sibling::* [3]
- This example selects the second preceding sibling of the context node, that is an element:
  sibling::* [-2]

Note:

By definition:

sibling::* [$pos]

when $pos > 0 is a shorthand for:

following-sibling::* [$pos]

and when $pos < 0 is a shorthand for:

preceding-sibling::* [-$pos]

- This example selects the innermost div ancestor of the context node:
  ancestor::div[1]
- This example selects the outermost div ancestor of the context node:
  ancestor::div[last()]
- This example selects the names of all the ancestor elements of the context node that have an @id attribute, outermost element first:
  ancestor::*[@id]

Note:

The expression ancestor::div[1] parses as an AxisStep with a reverse axis, and the position 1 therefore refers to the first ancestor div in reverse document order, that is, the innermost `div`. By contrast, `(ancestor::div)[1]` parses as a FilterExpr, and therefore returns the first qualifying `div` element in the order of the `ancestor::div` expression, which is in document order.

The fact that a reverse-axis step assigns context positions in reverse document order for the purpose of evaluating predicates does not alter the fact that the final result of the step is always in document order.

The expression `ancestor::(div1|div2)[1]` does not have the same meaning as `(ancestor::div1|ancestor::div2)[1]`. In the first expression, the predicate `[1]` is within a step that uses a reverse axis, so nodes are counted in reverse document order. In the second expression, the predicate applies to the result of a union expression, so nodes are counted in document order.
When the context value for evaluation of a step includes multiple nodes, the step is evaluated separately for each of those nodes, and the results are combined without reordering. This means, for example, that if the context value contains three `list` nodes, and each of those nodes has multiple `item` children, then the step `item[1]` will deliver a sequence of three `item` elements, namely the first `item` from each `list`, retaining the order of the respective `list` elements.