XPath and XPointer

CHAPTER

4

IN THIS CHAPTER

The XPath Recommendation describes a language for specifying sets of elements in an XML document. The XPath Recommendation provides the foundation for the XPointer Recommendation, *XML Pointer Language (XPointer)*, which extends the XPath language to allow for specifying any part of an XML document. XPointers specifically support incorporating fragment identifiers into URI references. The XPointer Recommendation can be found at `http://www.w3.org/TR/xptr/`.

XML schemas use a subset of the XPath location paths for identity constraints. The XPaths specified by identity constraints (see Chapter 13) serve to locate nodes (specifically elements and attributes) within a corresponding XML instance. An XML validator examines the values of these elements and attributes to ensure uniqueness or referential integrity.

An XPointer provides the location of an entire XML schema document or merely a set of schema components. An XML validator nominally assembles an entire XML schema from a collection of XML schema documents. In fact, any XML schema document may assemble individual components from a variety of XML documents (although they do not have to be XML schema documents) by using XPointers. Specifically, an XPointer can point to a set of schema components embedded in a non-schema XML document.

This chapter is not a comprehensive tutorial on XPath or XPointer. The respective Recommendations provide lots of detail and many examples. The book presents a more complete XPath tutorial—restricted to XPath usage in identity constraints—in Chapter 13. Section 4.2 presents a number of representative examples that demonstrate locating elements that represent schema components.

4.1 XPath

Fundamentally, an XPath is an *expression*. Evaluating an XPath expression results in one of the following:

- A node set
- A Boolean
- A floating-point number
- A string of Unicode characters

The scope of XPath is far more extensive than the constructs covered in this section. For example, many constructs are designed to support XSLT. This section covers only those portions of XPath pertinent to XML schemas. Specifically, identity constraints require the resultant node set to contain only elements or attributes. Fragment identifiers (see Section 4.2) further restrict the resultant node set to contain only elements. Therefore, the Boolean, floating-point number, and string values are mostly not relevant in the context of XML schemas (see Section 4.1.4 for a discussion of the exceptions).

Because the remainder of this chapter primarily discusses node set results, subsequent discussion focuses on the subset of an expression known as a *location path*. A location path always identifies a node set.

4.1.1 XPath Location Paths

Location paths nominally provide the grammar for typical XPath expressions for XML schemas. In an XML schema, all location paths are either relative to an enclosing component (for identity constraints) or relative to an entire XML document (for locating schema components). One of the general features of a location path is the ability to navigate along a number of axes. An *axis* specifies a direction of movement in the node tree. For example, you might specify a *child* node, an *attribute* node, an *ancestor* node, or a *descendant* node.

The XPath Recommendation defines 13 axes. An identity constraint is limited to containing only the axes *child*, *attribute*, and *descendant-or-self*. Furthermore, an identity constraint can only use the shortcut notation for these axes. Table 4.1 portrays these shortcuts (Table 4.3 contains the entire list of axes).

One schema document can locate another (or even parts of another) with an XPointer. An XPointer may specify any of the 13 axes in a predicate (see Sections 4.1.2 and 4.2 for more detail). Although technically feasible, not all the axes are particularly valuable with respect to schema document locations. Table 4.3 illuminates the axes with dubious merit.

4.1.2 Predicates

Predicates are very powerful, but slightly confusing when first encountered. A predicate is strictly a filter. A predicate filters out desired nodes from a node set.

TIP

In an XML schema, an XPath expression always results in a node set. An expression with a predicate also results in a node set. Specifically, an expression with a predicate results in a *subset* of the node set that corresponds to the same expression without the predicate.

The easiest way to demonstrate a predicate is to discuss two similar expressions along multiple axes. For demonstration purposes, consider a catalog consisting of several parts:

```
<catalog>
 <partNumber SKU="S1234">P1234</partNumber>
 <partNumber>P2222</partNumber>
 <partNumber SKU="Sabcd">Pabcd</partNumber>
</catalog>
```

The ensuing example returns a node set that contains all `partNumber` elements in a document:

```
//partNumber
```

The corresponding node set contains the `partNumber` elements whose values are '`P1234`', '`P2222`', and '`Pabcd`'.

An addition to the location path along an *attribute* axis results in a node set that contains all `SKU` attributes of `partNumber` elements:

```
//partNumber/@SKU
```

The corresponding node set contains the `SKU` attributes whose values are '`S1234`' and '`Sabcd`'.

The predicate in the next example, on the other hand, results in a subset of the `//partNumber` node set. In particular, the subsequent example returns `partNumber` elements that *have* `SKU` attributes, *not the attributes* as in the previous example.

```
//partNumber[@SKU]
```

The corresponding node set contains the `partNumber` elements whose values are '`P1234`' and '`Pabcd`'. The `partNumber` element whose value is '`P2222`' is *not* included, because this element has no `SKU` attribute.

An XPath expression, in general, can return any node set, a Boolean value, a string, or a floating-point number. In an XML schema, the results of an XPath expression (in an identity constraint), as well as the results of an XPointer expression (in a `schemaLocation` attribute value), must result in a node set. Because of this, this chapter does not go into detail describing the other result types. However, a predicate can refine a node set in an XPointer expression. Therefore, Table 4.4, which appears later in this chapter, provides a few examples of XPointers that contain predicates that return values that are not node sets. Because very few schemas contain XPointers, and even fewer contain complex XPointers, this chapter does *not* provide a comprehensive tutorial on predicates.

TIP

The Schema Recommendation does not support the use of predicates within identity constraints. Within an XML schema, predicates are valid only as part of an XPointer expression, which can be used only in conjunction with schema document (and part-of-document) locations.

4.1.3 Node IDs

An XPath expression may include any number of "functions" that either extract information from an XML document or help to restrict the resultant node set (via a predicate). Identity constraints

may not contain functions. In practice, XPath pointers to (parts of) schema documents rarely contain functions. Therefore, this chapter does not include a tutorial on the XPath functions or the corresponding return types. There is one function, however, supported—and even promoted by the XPointer Recommendation—in an XPointer location path expression: the `id` function.

The XPath `id` function is a great way to locate a specific node, assuming that a node specifies an ID. Note that the XPointer Recommendation surmises that IDs are "most likely to survive document change." The thought is that the structure of the XML document might change: Elements might "move" relative to one another. Consequently, a location path may no longer find the desired element, whereas the ID of the node is much more likely to be stable.

NOTE

The `id` function is part of the XPath Recommendation, which is why the discussion about this function appears in Section 4.1. With respect to XML schemas, however, an XPointer (in a `schemaLocation` value) is the only expression that can access this function. The `id` function *cannot* express any part of an identity constraint.

To demonstrate the use of the `id` function, the following URI locates the `catalogEntryDescriptionType` complex type (whose ID is '`catalogEntryDescriptionType.catalog.cType`') in the thematic catalog schema:

```
http://www.XMLSchemaReference.com/theme/catalog.xsd#xpointer
    (id("catalogEntryDescriptionType.catalog.cType"))
```

To encourage the use of IDs, the XPointer Recommendation permits an XPointer to consist of a shortcut, which is just the *bare name* of the ID:

```
http://www.XMLSchemaReference.com/theme/
    catalog.xsd#catalogEntryDescriptionType.catalog.cType
```

4.1.4 Using XPath with Identity Constraints

An identity constraint may reference only three of the axes specified by the XPath Recommendation: *child*, *attribute*, and *descendant-or-self*. Furthermore, an identity constraint may refer to these axes only via a shortcut. Table 4.1 lists all the shortcuts available (the three axes and a wildcard) in an identity constraint.

Table 4.2 provides a few examples that demonstrate how to locate elements in an XML document. Chapter 13, "Identity Constraints," provides a much more complete discussion.

Listing 13.3 covers the complete grammar for location paths, as pertains to identity constraints. Likewise, Table 13.2 provides a number of detailed XPath examples.

TABLE 4.1 Identity Constraint Shortcuts

Shortcut	*Meaning*	*Example*
(No axis following '/')	Implied *child* axis	`/a/b`
		Selects a `b` that is a subelement of an `a`, which is a subelement of the document root.
`@`	*attribute* axis	`a/@b`
		The `b` attribute of the `a` subelement element of the current node.
`//`	*descendant-or-self* axis	`a//b`
		Any b element that is a descendant of `a`.
`*`	A wildcard representing all immediate subelements	`a/*/b` Any b that is a subelement of any immediate subelement of a.

TABLE 4.2 Identity Constraint Examples

Example	*Meaning*
`//bulked`	This expression locates all `bulkID` elements along the *descendant-or-self* axis of the root element. Therefore, this locates all `bulkID` elements in the entire document.
`//bulkID/description`	The expression locates all `description` elements along the *child* axis of the previous example. Therefore, this locates all `description` elements that are immediate descendants of any `bulkID` element.
`/catalog/*/partNumber`	This expression locates all `partNumber` elements that are direct descendants of any (hence the '`*`') direct descendant of `catalog`. Note that each element reference is along the *child* axis.
`/catalog//partNumber`	This expression locates all `partNumber` elements that are descendants (but not necessarily *immediate* descendants) of `catalog`.
`/catalog//@employeeAuthorization`	This expression selects all `employeeAuthorization` attributes of *any* descendant of the `catalog` element.

4.2 XPointer

XML schema documents use `schemaLocation` attributes to locate other schema documents and parts of schema documents. The value of a `schemaLocation` is always a URI, which may include an XPointer. The `schemaLocation` attribute type of `schema`, `import`, and `include` are examples of where an XPointer might locate a schema document location.

An XPointer is nominally an extension of an XPath. The XPointer Recommendation permits—even encourages—the use of the XPath `id` function. There are also several XPointer specific extensions to XPath.

The XPointer Recommendation specifies expressions for returning portions of an XML document. The expression may evaluate to a node, a set of nodes, a portion of a node, or a portion of an XML document that spans nodes. Just as XML Schema limits the use of XPath expressions, it also limits the use of XPointer expressions. In particular, an XML schema may only import individual components (nodes) or sets of components (node sets). Because a component corresponds to an entire XML element—as opposed to a portion thereof—this section covers only XPointer constructs pertinent to extracting complete nodes.

4.2.1 Location Sets

The previous section mentions that an XPointer expression may evaluate to a node, a set of nodes, a portion of a node, or a portion of an XML document that spans nodes. Unlike an XPath expression, which must evaluate to a node set, an XPointer expression may theoretically return results that do not conform to a node. Therefore, the XPointer infrastructure requires an XPointer to return a *location set*.

A location set is an extension of a node set that an XPath normally returns. Each location in a location set is either a point or a range. A point consists of a node and an index. The index is a character offset into the node. A range consists of two points. The concepts of both point and range exist because of the XPointer requirement that an expression might return a subset of a node or possibly a set of characters that spans nodes.

Because an XML schema can only make use of an entire node or set of nodes, the remainder of this chapter covers only the subset of location information (and location sets) specified by nodes (and node sets). The terms ‘node set’ and ‘location set’ have similar meanings in the context of XML Schema.

4.2.2 Namespaces

The XPointer notation permits the identification of one or more namespaces. An XPointer expression specifies a namespace with the `xmlns` function. The argument to this function resembles a namespace attribute applicable to any XML element:

```
xmlns(xsd="http://www.w3.org/2001/XMLSchema")
```

The following XPointer locates the `catalogEntryDescriptionType` complex type by name. Namespace declarations always precede the locating expression:

```
http://www.XMLSchemaReference.com/theme/catalog.xsd#
  xmlns(xsd="http://www.w3.org/2001/XMLSchema")
  xpointer(
    xsd:schema/xsd:element[@name="catalogEntryDescriptionType"]
            )
```

In general, an XPointer may specify any number of namespaces. The previous example suffices for locating most schema components. An XPointer reference in a `schemaLocation` may require multiple namespaces when an XML document that is *not* a schema document has embedded Schema elements.

4.2.3 Subelement Sequences

Subelement sequences are notations that provide a shortcut to elements in an XML document. A subelement sequence has the following grammar:

```
bareName? ('/' [1-9] [0-9]*)+
```

where the optional *bareName* is replaced by the ID name of an element, as discussed in Section 4.2.2. The numerals represent the *N*th subelement (counting from 1) at each level.

WARNING

Subelement sequences provide an extremely compact and convenient short notation. Unfortunately, this supported notation is highly susceptible to failure. In particular, the structure of the expected XML document (in this case, most likely an XML schema document) must be extremely stable. Any change in the element structure of the document can provide surprising results.

The following example locates the fourth subelement of the third subelement of the document root:

```
../some.xsd#/1/3/4
```

Similarly, the following example locates the fourth subelement of the third subelement of the element whose ID is ‘`yadayada`’:

```
../some.xsd#yadayada/3/4
```

4.2.4 XPointer Extensions to XPath

This section covers *only* those XPointer extensions applicable to XML schemas. In fact, only one extension—the `range-to` function—has any applicability with respect to XML schemas. The use of this function is not common.

The XPointer Recommendation adds the `range-to` function as an option for an XPath step. A *step* is an axis and a node. A convenient use of the `range-to` function is to locate a set of nodes to incorporate into a schema. The `range-to` function is likely to appear in conjunction with the XPath `id` function. The following example locates four schema components that appear in sequence in `pricing.xsd`: `fullPriceType`, `freePriceType`, `salePriceType`, and `clearancePriceType`.

```
http://www.XMLSchemaReference.com/theme/pricing.xsd#
    xpointer(id("fullPriceType.pricing.cType")/
        range-to(id("clearancePriceType.pricing.cType")))
```

4.2.5 Using XPointer and XPath to Locate Schemas

This section describes the portions of the XPath Recommendation that apply to XPointers. An XPointer may reference any of the XPath axes touched on in Section 4.1.1. Table 4.3 lists all the axes supported by the XPath Recommendation and notes where each axis applies with respect to schema document locations. Such an XPointer might contain a reference to any axis; however, validations of many of these axes are likely to fail. Table 4.3 duly notes these likely failures in the Caveats column. A '✓' indicates that an XPointer can reference the corresponding axis and expect positive results.

An XPath expression, in general, can return any node set, a Boolean value, a string, or a floating-point number. In an XML schema, the results of an XPath expression (in an identity constraint), as well as the results of an XPointer expression (in a `schemaLocation` value), must result in a node set. Because both expressions return a node set, this chapter does not go into detail describing the other result types (Boolean, string, and number). However, a predicate can refine a node set in an XPointer expression. Therefore, Table 4.4 provides a few examples of XPointers that contain predicates, which return values that are not node sets. This chapter does not provide a comprehensive tutorial on the full XPath expression options that may appear in a predicate. Note that each example is only the XPointer part of a URI. An entire URI that includes an XPointer has the following form:

```
http://www.example.com/some.xml#xpointer(exampleXPointer)
```

See Sections 4.1.3 or 4.2.4 for examples of complete URIs. The cells in the Example column of Table 4.4 provide a substitution for *`exampleXPointer`* in the previous code excerpt.

Table 4.4 provides a nice illustration of the power of XPointer expressions enhanced by XPath predicates. For a complete tutorial on predicates, refer to the XPath Recommendation.

TABLE 4.3 XPath Axes Potentially Used in `schemaDocument` References

Axis	*Meaning*	*Caveats*
child	All subelements of the context node	✓
descendant	Element descendants of the context node	✓
parent	The parent of the context node	✓
ancestor	Element ancestors of the context node	An XPointer referencing this axis is technically okay, but the schema is probably bizarre at best.
following-sibling	All siblings of the context node that appear after the context node	Not recommended, as this will include attribute and namespace nodes (that is, undesirable nodes that are just portions of an element). Use the *following* axis instead.
preceding-sibling	All siblings of the context node that appear before the context node	Not recommended, as this will include attribute and namespace nodes (that is, undesirable nodes that are just portions of an element). Use the *preceding* axis instead.
following	Element siblings of the context node that appear after the context node	✓
preceding	Element siblings of the context node that appear before the context node	✓
attribute	Attributes of the context node	Not recommended: If the XPointer returns anything, the validation will fail.
namespace	Namespace nodes of the context node	Not recommended: If the XPointer returns anything, the validation will fail.
self	The context node	Why bother in an XPointer?
descendant-or-self	Descendant or self axes	✓
ancestor-or-self	Ancestor or self axes	✓

TABLE 4.4 XPointer Examples

Example	*Meaning*
id("foo")	The element whose ID is '`foo`' (see Section 4.1.3 for a detailed example).
id("foo")/range-to(id("bar"))	The elements whose IDs are '`foo`' and '`bar`', as well as all elements in between.
X[position()=1] or x[1]	The first `x`.
X[position()>1]	All `x` elements *except* the first one.
X[5]	The fifth `x`.
X[last()]	The last `x`.
./x	The `x` elements that are subelements of the parent of the current context.
X[y="hello"]	The `x` elements that have a subelement `y` whose value is '`hello`'.
X[@y and @z]	The `x` elements that have both `y` and `z` attributes.
X[@y or @z]	The `x` elements that have either a `y` or `z` attribute.
X[@y < "10"]	The `x` elements that have a `y` attribute whose value is less than '`10`'. Note that XPath supports '`=`', '`!=`', '`<`', '`<=`', '`>`', and '`>=`'. Escape these as necessary with the standard XML entity reference, such as '`<`' for the less-than character.
X[starts_with(y,"abc")]	The `x` elements that contain a `y` subelement whose value starts with '`abc`'.
X[sum(y) > 50]	The `x` elements that contain one or more `y` subelements, the sum of whose values is greater than 50.