窗体顶端

[PRODUCTS](https://www.progress.com/products)

**Share**

# FINDING XML NODES: PATH EXPRESSIONS

Just as SQL needs to be able to access any row or column in a relational table, XQuery needs to be able to access any node in an XML document. XML structures have both hierarchy and sequence, and can contain complex structure. Path expressions directly support hierarchy and sequence, and allow you to navigate any XML structure.

In this section, we discuss path expressions using an XML document, and then show path expressions used on an XML view of a relational table.

## **Path Expressions for XML Sources**

Let's explore path expressions using the following XML document.

<?xml version="1.0"?>

<portfolio id="Maggie">

<name>

<first>Maggie</first>

<last>Pristera</last>

</name>

<stocks>

<stock>

<ticker>AMZN</ticker>

<shares>3000</shares>

</stock>

<stock>

<ticker>EBAY</ticker>

<shares>4000</shares>

</stock>

<stock>

<ticker>IBM</ticker>

<shares>2500</shares>

</stock>

<stock>

<ticker>PRGS</ticker>

<shares>23</shares>

</stock>

</stocks>

</portfolio>

fn:doc() returns a document. The following example shows how to use fn:doc() with an absolute URL.

doc("file:///c:/data/xml/portfolio.xml")

The following example shows how to use fn:doc() with a relative URL.

doc("portfolio.xml")

By setting the Base URI, you can set the directory that is used to resolve relative URLs.

declare base-uri "file:///c:/data/xml/";

doc("portfolio.xml")

A path expression consists of a series of one or more "steps", separated by a slash (/ ) or double slash (//). Every step evaluates to a sequence of nodes. For example, consider the expression:

doc("portfolio.xml")/portfolio/name

The first step, doc("portfolio.xml"), returns a document node that represents the portfolio document.

The second step, portfolio, is a name test that specifies the name of an element; it returns the portfolio element at the top of the document, which is a child of the document node.

The third step, name, returns the element named "name", which is a child of the portfolio element.

Here is the result of the preceding query expression.

<name>

<first>Maggie</first>

<last>Pristera</last>

</name>

If a name test is preceded by the @ character, the name test matches an attribute rather than an element. For example, the expression doc("portfolio.xml")/portfolio/@id returns the id attribute of the portfolio element.

The double slash (//) allows steps to operate on any descendant of a node. For example, the expression doc("portfolio.xml")//name matches any element named name, anywhere in the portfolio document.

A predicate can be added to a step to set conditions for matching nodes. Predicates often set a condition on the children of a node. For example, the following path matches stock elements that contain a ticker element with the value "AMZN".

doc("portfolio.xml")//stock[ticker='AMZN']

Using the sample data, this expression produces the following result:

<stock>

<ticker>AMZN</ticker>

<shares>3000</shares>

</stock>

Conditions in a predicate can be combined using "and" and "or", as in the following expression.

doc("portfolio.xml")//stock[ticker='AMZN' or ticker='EBAY']

Conditions can be negated using fn:not(); for example, the following expression matches stock elements that do not have a ticker element with the value "AMZN":

doc("portfolio.xml")//stock[not(ticker='AMZN')]

One type of predicate is a numeric predicate, which sets a condition on the position of a node in a sequence. For example, the following expression finds the first stock element in a portfolio.

doc("portfolio.xml")//stocks/stock[1]

To understand how numeric predicates work in XQuery, you must know how XQuery evaluates a slash (/), as described in the following steps:

1. The expression on the left side of a slash is evaluated to produce a sequence of nodes.
2. The expression on the right side is evaluated for each context node drawn from the expression on the left side, and the results are combined.
3. When the numeric predicate is evaluated, it is evaluated for a given context node.

For example, in the preceding expression, when the numeric predicate is evaluated, the context node is a stocks element, the name test stock evaluates to a sequence of stock elements, and the numeric predicate matches the first stock in this sequence.

The following expression matches the first ticker element on each stock element.

doc("portfolio.xml")//stock/ticker[1]

To get the first ticker element in the document, use parentheses to make the expression on the left of the numeric predicate evaluate to the sequence of all ticker elements in the document.

(doc("portfolio.xml")//stock/ticker)[1]

## **Path Expressions for Relational Sources**

When XQuery is used to query relational data, relational tables are treated as though they are XML documents, and path expressions work the same way as they do for XML. Because relational tables have a simple structure, path expressions used for tables are usually simple.

No standard way exists to access a relational table in XQuery, so each XQuery implementation has its own way of doing this. In DataDirect XQuery, we use fn:collection() to access a relational table. For example, the following expression accesses the holdings table.

collection('holdings')

Each XQuery implementation must also decide how to map relational tables into XML in the XML view. The SQL 2003 standard has defined a standard set of mappings for this purpose as part of SQL/XML. Here is a SQL/XML mapping of the holdings table; this mapping represents each row as a holdings element, and represents each column of the table (userid, stockticker, shares) as an element that is a child of the holdings element.

<holdings>

<userid>Jonathan</userid>

<stockticker>AMZN</stockticker>

<shares>3000</shares>

</holdings>

...

<holdings>

<userid>Minollo</userid>

<stockticker>AMZN</stockticker>

<shares>3000</shares>

</holdings>

...

Once you understand the structure of the XML view, you can easily see how path expressions are applied to it. For example, the following expression finds holdings for the user whose userid is "Minollo".

collection('holdings')/holdings[userid='Minollo']

Because relational data is queried as if it were XML, some people think that relational tables are actually extracted from the database, turned into XML documents, and then queried, but this would be very inefficient. To the user, DataDirect XQuery makes all data look like XML, but to a SQL database, the implementation speaks SQL. Before evaluating the preceding expression, DataDirect XQuery converts it to a SQL expression similar to this one:

SELECT userid, stockticker, shares

FROM holdings

WHERE userid='Minollo'

|  |  |
| --- | --- |
|  | [Next:](https://www.progress.com/xquery/resources/tutorials/learning-xquery/using-xquery/xml-constructors)**["XML Constructors"](https://www.progress.com/xquery/resources/tutorials/learning-xquery/using-xquery/xml-constructors)** |

<https://www.progress.com/xquery/resources/tutorials/using-xquery/path-expressions>