software AG

Tamino

X-Query User Guide

Version 9.5 SP1

November 2013

tamino XML Server

This document applies to Tamino Version 9.5 SP1.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: INS-XQL-95SP1-20131030

Table of Contents

Preface

This document describes X-Query, one of the XML query languages available with Tamino. It focuses on the description of the language, how you use it for performing queries on XML objects, and how it differs from XPath.

This document, which has introductory character, explains the fundamentals of X-Query and includes some examples and pointers to related information.

1 What is X-Query?

X-Query is based on the **W3C**'s **XPath 1.0 specification**. XPath provides a data model and expression syntax for addressing parts of XML documents. X-Query adheres to the principles of XPath for efficient usage in the database context, whilst at the same time following the Tamino design principle of adhering to public standards.

XPath is not a query language in itself, but rather a language for addressing parts of an XML document. Since Tamino generally stores data as XML documents, XPath provides a standard mechanism for addressing the contents of XML objects returned from the database. With X-Query you can retrieve XML objects using:

- query expressions based on the XPath specification.

  These expressions are described in the section Expressions. In order to use these expressions effectively, you should be familiar with the current XPath specification.

- Tamino text retrieval and sorting expressions.

  Tamino also provides text retrieval and sorting expressions. You can retrieve text by using the "contains" operator ~=. You can sort XML objects by using the expressions `sortby` and `sortall`. These expressions are described in the X-Query Reference Guide.

- Tamino server extensions.

  You can extend X-Query by using Tamino server extensions. The documentation of X-Tension: Tamino Server Extensions describes how you can implement your own X-Query extension functions.

2 XPath 1.0 and X-Query

This chapter briefly explains the concepts of XPath, which is the basis of the X-Query language. It then outlines the relationship between XPath 1.0 and X-Query.

XPath 1.0 in a Nutshell

You can use XPath to address parts of an XML document. XPath is also the basis for XML-related languages such as **XSLT**. In XPath, an XML document is regarded as a tree in document order (i.e. depth-first) containing seven different types of nodes. These node types are:

root node;
element node;
text node;
attribute node;
namespace node;
processing instruction node;
comment node.

You can address nodes of any type and number with the help of XPath expressions. The details of this data model are described in the section **Data Model** of the XPath specification.

The expression syntax of XPath includes location paths for addressing tree nodes, and function calls of a core library used for working with strings, numbers and booleans. A location path starts from either the document root node (absolute path) or the context node (relative path) and has one or more location steps. A location step consists of three parts:

- an axis, which specifies the relationship between the set of selected nodes and the context node;
- a node test, which specifies the type and name of the set of selected nodes; and
- optional predicates, which further restrict the set of selected nodes.

There are thirteen axis directions, originating from the context node. The axis determines the initial node set, which is further refined by node tests and predicates. In XPath, you can specify a location path in either unabbreviated or abbreviated syntax. The following table lists the axes along with their direction (normal document order or reverse document order) and a short description. In the unabbreviated syntax a double colon '::' follows the name of the axis.

Axis	Direction	Meaning
`ancestor::`	reverse	The parent node and its ancestors up to the root node
`ancestor-or-self::`	reverse	The current node and its ancestors up to the root node
`attribute::`	implementation-defined	All attached attribute nodes
`child::`	normal	The immediate child nodes (default axis)
`descendant::`	normal	All descendant child nodes
`descendant-or-self::`	normal	The current node and all its descendant child nodes

Axis	Direction	Meaning
`following::`	normal	All nodes after the context node, excluding descendant nodes, attribute nodes and namespace nodes
`following-sibling::`	normal	All following nodes that are siblings of the current node
`namespace::`	implementation-defined	All attached namespace nodes
`parent::`	normal	The parent node (or attaching node for attribute and namespace nodes)
`preceding::`	reverse	All nodes before the context node, excluding descendant nodes, attribute nodes and namespace nodes
`preceding-sibling::`	reverse	All preceding nodes that are siblings of the current node
`self::`	normal	The current node

The node test determines the type and optionally the name of each node along the axis direction that is selected. For each axis, there is a principal node type: for the attribute axis, it is attribute; for the namespace axis, it is namespace; for other axes, it is element. You can select a node by applying one of the following node tests. The node is selected if the test evaluates to "true".

NodeTest	Description
`processing-instruction()`	A processing instruction node (regardless of name)
`comment()`	A comment node
`text()`	A text node
`node()`	A node of any type (regardless of name)
`processing-instruction('Literal')`	A processing instruction node with name `Literal`; if name is omitted, then the test is "true" for any processing instruction node
`'Name'`	A node of the principal node type
`'prefix:name'`	According to the axis used: an element node in the specified namespace, an attribute node in the specified namespace, or an empty node-set when using the namespace axis
`'*'`	According to the axis used: all element nodes, all attribute nodes or all namespace nodes
`'prefix:*'`	According to the axis used: all element nodes in the specified namespace, all attribute nodes in the specified namespace, or an empty node-set when using the namespace axis

The abbreviated syntax is as follows:

Abbreviation	Description
no axis	Nodes along the `child::` axis satisfying node tests and optional predicates
`@`	Nodes along the `attribute::` axis satisfying node tests and optional predicates
`.`	The `self::node()`, which is the current node of any type
`..`	The `parent::node()`, which is the empty node-set if the current node is the root node; the attaching node if the current node is an attached node (of type attribute or namespace); otherwise the parent node
`//`	`/descendant-or-self::node()/` At the start of an expression, this denotes the absolute location path; elsewhere, it denotes the relative location path

The last, optional part of a location step is a predicate to further restrict the set of selected nodes according to a qualifying expression. This expression is enclosed in square brackets `[` and `]`. The resulting nodes are ordered according to the direction of the selected axis. If instead you use a node-set expression, then the nodes are ordered in document order. You can use three different types of values in a predicate expression:

- A numeric value such as `patient[2]` or `patient[last()]`. The value is the proximity position of the node in the set, beginning with 1 for the originating context node.
- A node-set expression such as `medication[type]` or `type[@brand]`. This predicate is true if the node set returned is not empty.
- An expression such as `medication[count(type)>2]`. This predicate is true only if the expression is true.

See the **XPath specification** for more details. You can find examples based on the patient data set in the section **Querying XML Sample Documents** and in the reference section for the respective language elements.

From XPath to X-Query

This section documents the differences between expressions in X-Query and expressions in XPath. Since X-Query is based on the XPath specification, there are many more similarities than differences. However, X-Query differs from XPath in the following aspects:

- The following XPath functions are supported:

```
boolean
ceiling
count
false
floor
last
name
```

```
not
number
position
round
starts-with
string
sum
true
```

- The following XPath functions are currently not supported, but can be implemented using Tamino server extensions, as long as they do not have a variable number of arguments:

```
concat
contains
id
lang
local-name
namespace-uri
normalize-space
string-length
substring
substring-after
substring-before
translate
```

- X-Query supports the following additional operators that are not present in XPath:

```
adj
after
before
between
intersect
near
sortby
```

- X-Query has a binary “contains” operator ~= for text retrieval. There is no equivalent in XPath.
- X-Query does not support the unabbreviated syntax of location paths using named axes.
- X-Query does not support the use of variables.

XPath/X-Query and XML Schema

Historically, the development of XPath preceded the development of XML Schema; therefore, with a few exceptions, X-Query (which is closely related to XPath) is not related to XML Schema. In particular, an X-Query expression operates on string values or numeric values but not on typed values; in other words, an X-Query expression does not make use of the type information that is stored in the schema. For a strongly-typed view of the data, please consider using Tamino XQuery.

3 Querying XML Sample Documents

This chapter provides some examples of common queries using a small example Patient database. In the section Getting Started, you can find a step-by-step description of how to create and load this database in Tamino. To compare your query results with those presented here, load the data for the second patient as described in the section *Loading Objects into the Database*.

To execute a Tamino query, you can use the Tamino Interactive Interface, or you can enter an HTTP request directly in your browser's address line. If, for example, you have defined a database "mydb-test" on your local computer, and that database contains the collection "Hospital", which in turn contains the schema "patient", you could type the following URL in your browser's address line:

```
http://localhost/tamino/mydb-test/Hospital/patient?_XQL=/patient/name[surname~='At*']
```

to return all documents that belong to the schema 'patient' and satisfy the node-selection condition expressed by the query `/patient/name[surname~='At*']`.

Tamino queries can also be executed using any application environment that can handle HTTP requests, for example scripting languages. For more information, see the list of programming interfaces (APIs) in the Tamino documentation overview page.

The query response is returned by Tamino as an XML object. See the section Syntax of XML Responses in the X-Machine Programming documentation for a complete description.

The following table contains some typical query expressions for the Patient schema. The first column contains the query expression, the second column contains a description of the syntax used in the query, the third column describes the result of the query in general terms, and the fourth column is the query result based on the existing instances of the Patient schema.

Query Expression	Description	General Retrieval Result	Specific Retrieval Result
`/patient`	The path operator / at the beginning of a query indicates that selection starts at the root node.	All `patient` children of the root node.	2 `patient` nodes ("Atkins" and "Bloggs")
`//therapy`	The path operator // at the beginning of a query indicates selection of a node regardless of its location within the XML object.	All `therapy` descendants of the root node.	2 `therapy` nodes (for "Atkins" and "Bloggs")
`/patient/name/firstname`	The path operator / indicates the next level in the hierarchy.	All `firstname` nodes that are children of `name` nodes that in turn are children of a `patient` node.	2 `firstname` nodes ("Paul" and "Fred")
`/patient/therapy//type`	The path operator // indicates any number of intervening hierarchy levels.	All `type` nodes that are descendants of `therapy` nodes that in turn are children of a `patient` node.	1 `type` node (for "Bloggs")
`/patient[//surname='Atkins']`	Equality and relational operators (=, !=, <, >, <=, >=) can be used for comparison operations.	All `patient` nodes that have a descendant `surname` node with the content "Atkins".	1 `patient` node ("Atkins")
`//therapy/medication/type [@form = 'tablet']`	The predicate expression inside [] filters the set of nodes to its left based on the conditions inside the brackets. @ selects the attribute node attached to an element node.	All `type` nodes below a `medication` node whose attached attribute node `form` has the string value "tablet".	1 `type` node (for "Bloggs")
`/patient[born < 1960 and //city='Bradford']`	Boolean operators (`and`, `or`) can be used to express multiple conditions.	All `patient` nodes that have a value less than 1960 for the `born` element and a value of "Bradford" for any `city` element below the root node.	1 `patient` node ("Bloggs")

The following table contains some typical query expressions using X-Query's text retrieval and sorting expressions:

Query Expression	Description	General Retrieval Result	Specific Retrieval Result
`//occupation [.~='Professional']`	The contains operator ~= performs a selection based on word content. The specified word (case insensitive) is found irrespective of its location within the text node.	All `occupation` nodes that contain the word value "Professional".	1 `occupation` node (for "Atkins")
`/patient/name [surname~='At*']`	The contains operator ~= can be used to perform selection using the wildcard character *.	All `name` nodes that have a `surname` child node whose value contains a word beginning with "At".	1 `name` node (for "Atkins")
`//remarks after therapy`	The operators `before` and `after` can be used to perform selection based on sibling positioning.	All descendant `remarks` nodes that follow a `therapy` node.	1 `remarks` node (for "Bloggs")
`/patient[born between 1950,1953]`	The `between` operator can be used to perform selection based on a value range.	All `patient` nodes that have a `born` child node whose value is between 1950 and 1953.	1 `patient` node (for "Bloggs")
`/patient[occupation ~= 'professional' adj 'diver']`	The `adj` operator can be used to perform selection based on adjacent value location.	All `patient` nodes that have a child `occupation` node that contains a word "Professional" immediately followed by the value "diver".	1 `patient` node ("Atkins")
`/patient[occupation ~= 'professional' near 'diver']`	The `near` operator can be used to perform selection based on proximity value location.	All `patient` nodes that have an `occupation` element that contains a word "professional" immediately followed by the value "diver" (or vice versa).	1 `patient` node ("Atkins")
`//firstname sortby (.)`	`sortby` can be used to sort the query results to be sorted in a certain order.	All `firstname` elements sorted by `firstname` within each document.	The `firstname` elements of the first document ("Atkins") are returned in sorted order, followed by the sorted `firstname` elements of the second document ("Bloggs").

4 Related Information

The X-Query documentation focuses on X-Query as a language. However, since database queries are the principal means of communicating with the database server, you might have a problem that arose because of a certain query. If you cannot find the information you require in the X-Query documentation, please have a look at the following sources of information, all of which also deal with aspects of querying a Tamino database:

Internal Resources

- X-Query Reference Guide: This is the complete reference manual, describing all expressions and functions available in X-Query.
- How to Query a Database with Tamino X-Query using the Tamino Interactive Interface: This section describes how to query a database object using the Tamino Interactive Interface.
- Efficient Queries: X-Query: Here you can find some advice for speeding up your queries.
- How to write user-defined Query Functions as Tamino Server Extensions: You can use server extensions to extend the capabilities of X-Query. This section explains how to implement a server extension in Tamino.
- Querying Using X-Machine Commands: This section explains how to query a database directly using HTTP and X-Machine command verbs.
- Storing Non-XML Objects in Tamino: This section explains how to represent and query objects that are in some other format, for example Microsoft Word or Excel documents or PDF files.
- Unicode and Text Retrieval: This section discusses character handling and word recognition. If you use text retrieval in your queries, this section gives you detailed background information about how Tamino performs word-wise full-text operations.
- Character Encoding of XML objects: This section discusses the character encoding of XML objects when querying them directly using HTTP and X-Machine command verbs.

External Resources

- **The Namespaces in XML 1.0 specification**: This W3C recommendation defines basic language elements that are used in XPath.
- **The XPath Language Version 1.0 specification**: The official XPath specification of the W3C, which forms the basis of X-Query.
- *XSLT: Programmer's Reference*: This book by Michael Kay (published by **Wrox Press**) offers very detailed information about XPath and XSLT. The second edition covers aspects of XSLT 1.1. For a solid understanding of XPath, the first edition serves equally well.