TeXQuery: A Full-Text Search Extension to XQuery

Part I – Language Specification

Sihem Amer-Yahia [1]
sihem@research.att.com

Chavdar Botev [2]
cbotev@cs.cornell.edu

Jonathan Robie [3]
jonathan.robie@datadirect-technologies.com

Jayavel Shanmugasundaram [2]
jai@cs.cornell.edu

[1] AT&T Labs

[2] Computer Science Department, Cornell University

[3] DataDirect Technologies

14 July 2003

Contents

1. Introduction

TeXQuery is an extension to the XQuery language [3]. It provides full-text functionality using postfix full-text operators, which are easily composed with other XQuery expressions. Full-text conditions are expressed using a sublanguage with a syntax much like that of SQL/MM. This sublanguage is not placed in strings, but it occurs in a well-defined context as an operand of full-text operators. For instance, in the following example, ftcontains is a full-text operator that returns a Boolean value:

```
//book[title ftcontains ("heuristic evaluation" && "usability"
                         with stems])
```

The left-hand operand of ftcontains is an XQuery expression specifying the nodes to be searched. The right hand operand is an expression in the full-text sublanguage.

Here are some of the important features of TeXQuery:

- **Expressive power:** TeXQuery supports phrase matching, regular expressions, Boolean connectives, proximity, stemming, use of thesauri, and lexical control over term matching. The solutions to the full-text use cases [5] illustrate this in more detail.
- **Support for scored results:** TeXQuery provides a powerful mechanism for scoring full-text query results. Users can specify a general full-text expression as the scoring expression, and this can be different from the full-text search expression. In addition, users can express score weights in in the full-text scoring expression.
- **Composable full-text search primitives:** TeXQuery full-text search primitives are fully composable. Thus, arbitrarily complex full-text search expressions can be created by combining basic full-text search primitives.
- **Pair-wise composability with regular XQuery expressions:** TeXQuery full-text search expressions can be nested under arbitrary XQuery expressions, and vice-versa.
- **Basis in existing practice:** The TeXQuery full-text language is similar to existing full-text languages, and the relationship between structure and full-text is similar to that of full-text search in relational databases. The semantics of full-text operators is also quite conventional.
- **Easy integration with XQuery:** TeXQuery does not require any changes to the XQuery data model. It also does not require any changes to existing XQuery constructs.

1.1. Rationale for the sublanguage approach

XQuery is defined on XML structures. Full-text search operates on linguistic units such as words, sentences, and paragraphs, which are not represented in the XQuery 1.0 and XPath 2.0 Data Model. The basic operators for full-text search and the way these operators combine are also quite different from the operators of XQuery. For instance, the 'and' and 'or' operators of XQuery are Boolean operators with a well-defined meaning, but their full-text equivalents are operators on linguistic units whose semantics are quite different; further, these full-text operators have to interact with full-text predicates on term positions (such as textual proximity); therefore, this proposal introduces new operators for the full-text sublanguage. Even though full-text operators are not defined in terms of XML structures, the text on which they operate is generally found in XML instances, and it is vital that full-text be carefully integrated with the XQuery language.

Proposals for adding full-text to XQuery have taken a variety of syntactic approaches, representing full-text conditions using SQL/MM strings, composable functions, XML fragments, or operators added to the XQuery language. This proposal uses a small set of operators that use a

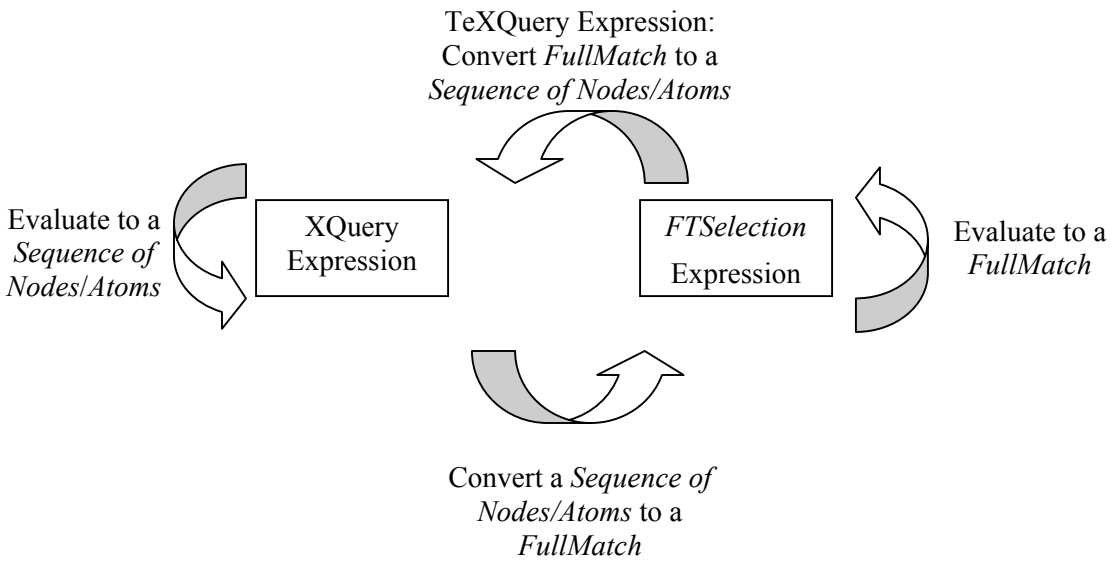

Figure 1 Composing XQuery and TeXQuery expressions

sublanguage for full-text search conditions. The full-text sublanguage is used only in a few well-defined contexts, and full-text operators can be composed flexibly with other XQuery expressions. Full-text operators can take operands that are normal XQuery expressions, and return results that can be used by XQuery expressions.

1.2. Composing XQuery and Full-Text Expressions

Full-text expressions can be nested inside regular XQuery expressions, and XQuery expressions occur in the productions of full-text expressions. Since the XQuery expressions do not define operations for linguistic units, and full-text operations are not defined in terms of XML structures such as elements and attributes, conversions between the two data models must be defined. This is shown in Figure 1

XQuery expressions take (possibly many) sequence of nodes/atoms as input and evaluate to a sequence of nodes/atoms (left arrow in Figure 1). In contrast, full-text search expressions – called *FTSelection*s in TeXQuery – take in (possibly many) *FullMatch*es as input, and evaluate to a *FullMatch* (right arrow in Figure 1). *FullMatch* is the data model underlying *FTSelection*s, and captures the notion of linguistic token positions, and other information required to make *FTSelection*s fully compositional. Specifically, a *StringMatch* represents a match for a given string of a full-text expression, a *SimpleMatch* represents a conjunction of *StringMatch* conditions (allowing negation), and a *FullMatc*h represents the entire result of a full-text query as a set of *SimpleMatch* components.

The conversion from the *FullMatch* data model to the XQuery data model is as follows. We propose three new XQuery expressions for full-text search, hereafter referred to as TeXQuery expressions, which convert a *FullMatch* to a sequence of nodes/atoms. This enables TeXQuery expressions to be nested and composed with regular XQuery expressions. This is shown as the top arrow in Figure 1.

The conversion from the XQuery data model to a *FullMatch* is done as follows. We use the result of an XQuery expression as a search token in an *FTSelection* by converting the XQuery expression to the *FullMatch* associated with that search token. For example, in the *FTSelection*

"`'XML' && article/title`", we will convert the string result of the "article/title" expression to a *FullMatch* corresponding to the string result (including linguistic token positions, etc.). This interaction between XQuery and *FTSelection*s is shown as the bottom arrow in Figure 1.

1.3. Outline

In the rest of this proposal, we provide an overview of the changes to the XQuery syntax and give a high-level description of the semantics of the new language elements. The formal semantics of the new sub-language and solutions to XQuery full-text use cases are available in supporting documents.

2. Terminology and Example Document

We use the following terminology in the description of TeXQuery.

Evaluation context	A sequence of nodes as defined in the XQuery data model that defines the context over which the full-text search is performed
Special character	Character used for specifying syntactic structure (e.g. marking the end of a sentence, delimiting a sentence clause, etc.)
Whitespace character	Space, tabulation or new line character
Linguistic token	A sequence of characters that corresponds to a token in a given human language. In Western languages and many other languages, a linguistic token corresponds to a word. We use the term 'linguistic token' rather than word because some languages may not have a construct that corresponds precisely to the Western notion of a word.
Search token	A sequence of characters defining a pattern for searching linguistic tokens
Search token occurrence	A linguistic token and/or its position that corresponds to the search token according to the search token matching rules

We use the following XML document fragment to illustrate examples in the language specification.

```
<book number="1">
   <title shortTitle="Improving Web Site Usability">Improving
      the Usability of a Web Site Through Expert Reviews and
      Usability Testing</title>
   <authors>
      <author>Millicent Marigold</author>
      <author>Montana Marigold</author>
   </authors>
   <editors>
      <editor>Véra Tüdor-Medina</editor>
   </editors>
   <content>
      <p>The usability of a Web site is how well the
         site supports the users in achieving specified
         goals. A Web site should facilitate learning,
         and enable efficient and effective task
         completion, while propagating few errors.</p>
      <note>This book has been approved by the Web Site
         Users Association.</note>
   </content>
</book>
```

3. TeXQuery Expressions

We introduce three new XQuery expressions for full-text search, hereafter referred to as the TeXQuery expressions. The three expressions are: *FTContainsExpr*, *FTScoreExpr,* and *FTSearchExpr*.

FTContainsExpr returns true if there is an XQuery node in the specified evaluation context that satisfies a given *FTSelection*. This roughtly corresponds to the *contains()* predicate in XQuery, but is much more powerful. *FTContainsExpr* will probably be the most frequently used TeXQuery expression, as can be seen in the solutions to the use cases.

FTScoreExpr returns the scores of a sequence of nodes with respect to an *FTSelection*. *FTScoreExpr* is a more powerful construct that *FTContainsExpr* and allows search results to be ordered by score, allows top-*k* results to be returned, etc.

FTSearchExpr is used for exploratory full-text search over an evaluation context. FTSearchExpr is more powerful than FTContainsExpr in the sense that users do not have to specify the types of the return nodes. This enables users to issue exploratory queries even if they are not aware of the underlying schema or structure of the underlying XML documents [1][2].

The TeXQuery expressions return results in the XQuery data model, i.e., they all return sequences of nodes/atoms. Consequently, TeXQuery expressions can be arbitrarily composed with other regular XQuery expressions.

The three TeXQuery expressions are described in more detail in the following sections.

3.1. *FTContainsExpr*

3.1.1. Description

FTContainsExpr returns a Boolean result specifying whether some node in the specified evaluation context satisfies a full-text search condition.

3.1.2. Syntax

FTContainsExpr ::= Expr "`ftcontains`" FTSelection

3.1.3. Semantics

Expr is an XQuery expression that is evaluates to a sequence of nodes; this sequence of nodes sets the evaluation context. The *FTContainsExpr* returns true if and only if some node in the evaluation context satisfies the *FTSelection*. *FTSelection* is defined in a later section.

3.1.4. Examples

The query

```
//book[.//author ftcontains "Marigold"]/title/@shortTitle
```

will return the short titles of the books that have Marigold as an author.

The query

```
//book[./title ftcontains "Web Site Usability"
       and
      .//author ftcontains "Marigold"]/@number
```

will return the numbers of the books containing the phrase "Web Site Usability" in their title and "Marigold" as an author.

3.2. *FTScoreExpr*

3.2.1. Description

FTScoreExpr returns a sequence of floating point numbers that represent the scores of the nodes in the specified evaluation context. The score is computed using a specified full-text selection condition.

3.2.2. Syntax

FTScoreExpr	::= Expr "`ftscore`" FTSelectionWithScoreWeights

3.2.3. Semantics

Expr is an XQuery expression that is evaluates to a sequence of nodes; this sequence of nodes sets the evaluation context. The *ScoreByExpr* returns a sequence of floating point numbers, each of which is a score for a node in the evaluation context computed using *FTSelectionWithScoreWeights*. The scores of the nodes are returned in the same order as the nodes in the evaluation context. *FTSelectionWithScoreWeights* is defined in a later section.

The actual scoring function is implementation-defined, but it should satisfy the following properties:

1. The scoring function should produce score values of type xs:float in the range [0, 1].
2. If a node in the evaluation context does not satisfy the *FTSelectionWithScoreWeights*, its score should be 0.
3. If a node in the evaluation context satisfies the *FTSelectionWithScoreWeights*, its score score should be > 0.
4. For nodes in the evaluation context that satisfy the *FTSelection*, a higher score should imply a higher degree of relevance with respect to *FTSelectionWithScoreWeights*.

3.2.4. Examples

The query

```
//book ftscore "web site" && "usability" && "testing"
```

will return the scores of all books that contain "web site", "usability" and "testing".

The following query will return the results of the above query ordered by their score:

```
for $item in Expr
let $score := $item
              ftscore
              "web site" && "usability" && "testing"
order by $score descending
return <hit>{$item}<score>{$score}</score></hit>
```

The following query will return the books containing the phrase "Web Site Usability" and order them by their score with regards to the query "user testing":

```
for $item in //book[. ftcontains "Web Site Usability"]
let $score := $item score by "user testing"
```

```
order by $score descending
return <hit>{$item}<score>{$score}</score></hit>
```

The following query will return the books about “Web Site Usability” with a score above a threshold:

```
for $item in //book
let $score := $item score by “Web Site Usability”
where $score > 0.75
order by $score descending
return <hit>{$item}<score>{$score}</score></hit>
```

The next query will return the top 10 books about “Web Site Usability”.

```
for $hit at $i in
   for $item in //book
   let $score := $item score by “Web Site Usability”
   order by $score descending
   return <hit>{$item}<score>{$score}</score></hit>
where $i <= 10
return $hit
```

3.3. *FTSearchExpr*

3.3.1. Description

FTSearchExpr evaluates a full-text search within a specified evaluation context, and returns the most specific result nodes in rank order.

3.3.2. Syntax

FTSearchExpr	::= Expr? "`ftsearch`" FTSelectionWithScoreWeights

3.3.3. Semantics

Expr is an XQuery expression that is evaluates to a sequence of nodes; this sequence of nodes sets the evaluation context. Only the nodes in the evaluation context or its descendants (referred to as the full evaluation context) are returned as the result of *FTSearchExpr*. If no *Expr* is specified, the current context sequence is used as an evaluation context.

FTSearchExpr returns the most specific nodes from the full evaluation context that satisfy *FTSelectionWithScoreWeights*. Each result node is scored with respect to *FTSelectionWithScoreWeights*, and the result sequence is ordered based on this score. The formal semantics document (Part II) formally defines the semantics of "most specific result node". Intuitively, this is a node that satisfies *FTSelectionWithScoreWeights* such that none of its descendants satisfy *FTSelectionWithScoreWeights*.

3.3.4. Examples

The query

```
/book[@number="1"] ftsearch "Millicent"
```

will return the first "author" element because they are the most specific elements, containing the search string.

Similarly, the query

```
/book[@number="1"] ftsearch "Marigold" && "Usability"
```

will return the "book" element because it is the most specific one containing both "Marigold" and "Usability".

Note that ftsearch dynamically decides the result granularity (author or book) based on query keywords, and is thus useful for exploratory XML keyword search [1][2].

4. *FTSelection*s

This section describes *FTSelection* and *FTSelectionWithScoreWeights* used in the definition of TeXQuery expressions. These represent the full-text search conditions, associated context modifiers, and score weights.

4.1. *FTSelection*

4.1.1. Description

The *FTSelection* production specifies all permitted kinds of full-text search conditions. The *FTSelectionWithScoreWeights* production is an *FTSelection* that also allows the specification of score weights.

4.1.2. Syntax

```
FTSelection                 ::= FTStringSelection |
                                FTAndConnective |
                                FTOrConnective |
                                FTNegation |
                                FTMildNegation |
                                FTOrderSelection |
                                FTScopeSelection |
                                FTDistanceSelection |
                                FTWindowSelection |
                                FTTimesSelection |
                                FTSelection FTContextModifier

FTSelectionWithScoreWeights ::= FTStringSelection |
                                FTAndConnective |
                                FTOrConnective |
                                FTNegation |
                                FTMildNegation |
                                FTOrderSelection |
                                FTScopeSelection |
                                FTDistanceSelection |
                                FTWindowSelection |
                                FTTimesSelection |
                                FTSelectionWithScoreWeights FTContextModifier |
                                FTSelectionWithScoreWeights "weight" xs:float
```

4.1.3. Semantics

FTSelection is an abstract construct and has no semantics on its own. It carries the semantics of the particular full-text search condition used.

The *FTContextModifier* defines the full-text search environment, and modifies the operational semantics of the *FTSelection* it is applied on. Examples of context modifiers include stemming, ignoring of stop-words, etc. A *FTContextModifier* only applies to the *FTSelection* (and nested *FTSelection*s) it is applied on. Outside this scope it has no influence on the valuation of the *FTSelection*. *FTContextModifier* is defined in Section 5.

The "`weight`" clause is used by the implementation-defined scoring function. The larger the weight associated with an *FTSelection*, the higher the score of nodes that satisfy this *FTSelection*.

4.1.4. Examples

See the examples for the various types of *FTSelection*s.

4.2. *FTStringSelection*

4.2.1. Description

This *FTSelection* specifies the linguistic tokens that must be contained by a result node.

4.2.2. Syntax

FTStringSelection	::= Expr (“`any`” | “`all`” | “`phrase`”)?

4.2.3. Semantics

Expr must evaluate to a sequence of string atomic values or nodes of type “xs:string”.

FTStringSelection specifies a bag of linguistic tokens that should be contained within a node.

The simplest case is when *Expr* evaluates to a singleton string sequence. If the special-characters context (see 5.4) is “`without special characters`”, any white-space and punctuation characters (except those participating in regular expression) are used to break the string into search tokens. If there are more then one search tokens, the string matching is equivalent to specifying them as a sequence with “`phrase`”.

If a search token consists of only alpha-numeric characters, it or linguistic tokens that can be derived using the current FTS context (5) must be contained in the result node.

Tokens may be grep-style regular expressions with the following differences :

– “`^`” is matched by the beginning of an element content;

– “`$`” is matched by the end.

– “.”is matched by any alpha-numeric or punctuation character

In this case, all the terms that match the regular expression are matched by *FTStringSelection*.

If the length of the result sequence of *Expr* is greater than one, then the *FTStringSelection* is processed as:

– *FTAndConnective,* which combines every string as a separate *FTStringSelection*, if “`all`” or nothing is specified (default).

– *FTOrConnective*, which combines every string as a separate *FTStringSelection*, if “`any`” is specified.

– *FTAndConnective,* which combines every string as a separate *FTStringSelection*, followed by ‘`ordered with word distance 0`’ (see 0 and 4.9) if the clause “`phrase`” is specified.

4.2.4. Examples

The query

```
/book[@number=”1”] ftcontains “Web Site Usability”
```

The above query will returns true because “Web Site Usability” is contained in the “shortTitle” attribute of the “title” element.

The expression

```
/book[@number=”1”]//p ftcontains “Web Site Usability”
```

will return false because the paragraph doesn't contain the phrase "Web Site Usability" though it contains all of the search tokens in the phrase.

The query

```
for $book in /book[.//author ftcontains "Marigold"]
let $score := $book/title score by "Web Site Usability"
where $score > 0.8
order by $score descending
return $book/@number
```

will return the most relevant books by Marigold with a title about "Web Site Usability" in sorted by score order.

The query

```
//* ftsearch ("Web Site" "Usability")
```

will return the most specific elements in the database that contain the phrase "Web Site" and "Usability".

4.3. *FTAndConnective*

4.3.1. Description

FTAndConnective is used to obtain the nodes that satisfy simultaneously two selection criteria.

4.3.2. Syntax

FTAndConnective	::= FTSelection "&&" FTSelection

4.3.3. Semantics

Any result node must satisfy both *FTSelection* criteria.

4.3.4. Examples

The query

```
/book[@number="1"]/title ftcontains ("usability" && "testing")
case insensitive
```

will return true because it contains "usability" and "testing" if we ignore the letter case.

The query

```
/book[@number="1"/authors ftcontains "Millicent" && "Montana"
```

will return true because they are contained in the "author" children elements.

The query

```
/book ftsearch "usability" && "testing"
```

will return the most specific descendants of book elements that contain the search tokens "usability" and "testing".

4.4. *FTOrConnective*

4.4.1. Description

FTOrConnective can be used to obtain the nodes that satisfy at least one of two selection criteria.

4.4.2. Syntax

FTOrConnective	::= FTSelection "||" FTSelection

4.4.3. Semantics

Any result node should satisfy at least one of the *FTSelection* criteria.

4.4.4. Examples

The query returning books written by "Millicent" or "Montana" can be written as:

```
/book[.//@author ftcontains "Millicent" || "Montana"]
```

The query

```
/book[@number="1"] ftsearch "Millicent" || "Montana"
```

will return the two "author" elements.

4.5. *FTNegation*

4.5.1. Description

FTNegation can be used to obtain nodes that do not satisfy a certain selection criterion.

4.5.2. Syntax

FTNegation	::= "!" FTSelection

4.5.3. Semantics

Result nodes must not satisfy the specified selection condition.

4.5.4. Examples

Books containing "information" and "retrieval" but not "information retrieval" are returned by:

```
/book[. ftcontains "information" && "retrieval" && ! "information
retrieval"]
```

Books about "web site usability" but not "usability testing" can be found using:

```
/book[. ftcontains "web site usability" && ! "usability testing"]
```

Consider the sample text fragment in section 2. The query

```
/book[@number="1"] ftsearch "usability" && ! "goal"
```

will return the "title" element but not the "p" element because it contains "goal" directly and not the "book" element because it contains "goal" indirectly.

4.6. *FTMildNegation*

4.6.1. Description

FTMildNegation can be used as a milder form of "*&& !*" where the exclusion from the result consideration is performed at the occurrence level and not at the node level.

4.6.2. Syntax

FTMildNegation	::= FTSelection "`mildnot`" FTSelection

4.6.3. Semantics

A node is considered a result if contains search token occurrences that satisfy the first condition and do not satisfy the second. Therefore, a node can be a result even if it contains occurrences of the second condition; the will be only excluded from the score computation.

4.6.4. Examples

Books about "web site usability" ignoring any reference to "usability testing" can be found using:

```
/book[. ftcontains "web site usability" mildnot "usability
testing"]
```

The query

```
/book[@number="1"] ftsearch "usability" mildnot "usability
testing"
```

will return the both the "title" and the "p" elements but the occurrence of "Usability Testing" in the former will not be considered when computing it score.

4.7. *FTOrderSelection*

4.7.1. Description

FTOrderSelection can be used to enforce that the order of search token occurrences in the matched node is the same as they are specified in the query.

4.7.2. Syntax

FTOrderSelection ::= FTSelection "`ordered`"

4.7.3. Semantics

By default, there are no restrictions on the order in which the query linguistic tokens are matched in the document.

The *FTOrderSelection* can be used to impose such an order. A node is considered a result if it matches the nested selection condition and all search token occurrences in it are in the order that they are specified in the query.

4.7.4. Examples

If we are interested only in the book title of books that contain "web site" and "usability" in this order, we can use:

```
/book[. ftcontains ("web site" && "usability") ordered]/title
```

Consider the sample text fragment in section 2.

The query expression

```
/book[@number="1"]/title ftcontains ("Montana" "Millicent")
ordered
```

will return false because the names do not appear in the query order.

The query

```
/book[@number="1"] ftsearch ("web site" && "usability") ordered
```

will return the "title" element; the "p" element will not be returned because its occurrences of "web site" and "usability" do not match the order in the query.

4.8. *FTScopeSelection*

4.8.1. Description

FTScopeSelection specifies a condition oh the scope of the occurrences of the matched search tokens.

4.8.2. Syntax

FTScopeSelection	::= FTSelection "`same`" ("`node`" | "`sentence`" | "`paragraph`") | FTSelection "`different`" ("`node`" | "`sentence`" | "`paragraph`")

4.8.3. Semantics

FTScopeSelection specifies whether any matched linguistic token in *FTSelection* should be directly contained in the same ('`same`') or different ('`different`') scope. Possible scopes include element node (from the XQuery data model), sentence (delimited by ".", "!", or "?"), or paragraph (delimited by blank lines and EOLN/CR characters).

By default, there are no restrictions on the scope of the occurrences, i.e. the occurrences may occur in any combination of nodes, sentences, and paragraphs.

4.8.4. Examples

The query

```
/book[@number="1"] ftcontains ("usability" && "Marigold") same
node
```

will not return the "book" element because "usability" and "Marigold" are not directly contained by the same element node. On the other hand,

```
/book[@number="1"] ftsearch ("usability" && "Marigold") different
node
```

will return the "book" element because it indirectly contains the linguistic tokens and they appear in different nodes.

Finding books mentioning "usability" and "testing" in the same paragraph can be achieved using:

```
/book[. ftcontains ("usability" && "testing") same paragraph]
```

4.9. *FTDistanceSelection*

4.9.1. Description

FTDistanceSelection allows control over the distance between every pair of matched linguistic tokens.

4.9.2. Syntax

FTDistanceSelection	::= FTSelection "`with`"? ("`word`" | "`sentence`" | "`paragraph`") "`distance`" FTRangeSpec
FTRangeSpec	::= ("`exactly`"? Expr) | ("`at least`" Expr) | ("`at most`" Expr) | ("`from`" Expr "`to`" Expr)

4.9.3. Semantics

FTRangeSpec specifies a range of integers.

The XQuery expression(s) *Expr* must evaluate (with atomization) to a singleton sequence with an integer atom. Otherwise, the expression containing the clause must return error.

Let the first XQuery expression *Expr* evaluates to *M* and the second XQuery expression in the last type of *FTRangeSpec* evaluates to *N*.

The format with "`exactly`" limits the range to a single integer: [*M*, *M*]. "`at least`" can be used to specify the range [*M*, ∞). The "`at most`" variant specifies the range [0, *M*].

The last variant can be used to specify a range of allowable values: the closed interval [*M*, *N*].

FTDistanceSelection can be used to limit the distance in number of linguistic tokens, sentences, or paragraphs between consecutive occurrences of the search tokens in *FTSelection.* These correspond to "`word distance`", "`sentence distance`", and "`paragraph distance`" forms of *FTDistanceSelection*.

Any linguistic tokens in the stop-words context (see 5.7) will not be counted against the word distance.

4.9.4. Examples

'`exactly 0`' specifies the range [0, 0].

'`at least 1`' specifies the range [1, ∞).

'`at most 1`' specifies the range [0, 1].

'`from 5 to 10`' specifies the range [5, 10].

We can search for containing "information" and "retrieval" but will discard those occurrences of the search tokens that are more than 10 linguistic tokens apart.

```
/book[. ftcontains ("information" && "retrieval) mildnot
(("information" && "retrieval") word distance at least 11)]
```

Finding the titles of books mentioning “web”, “site”, and “usability” with at most 2 intervening linguistic tokens between consecutive occurrences of the linguistic tokens can be achieved with:

```
/book[. ftcontains (“web” && “site” && “usability”) with word
distance at most 2]/title
```

Consider the sample text fragment in section 2. The query

```
/book[@number=”1”] ftsearch (“web site” && “usability”) with word
distance at most 1
```

will return the “title” element; the “p” element will not be returned when stop-words are not ignored because its occurrences of “web site” and “usability” are within word distance of 2.

4.10. *FTWindowSelection*

4.10.1. Description

FTWindowSelection allows control over the distance between the leftmost search token occurrence (the one with the smallest position) and the rightmost one.

4.10.2. Syntax

FTWindowSelection ::= FTSelection (“`within`”)? “`window`” FTRangeSpec

4.10.3. Semantics

The number of linguistic tokens for the occurrences of the nested selection condition between the smallest linguistic token position and the largest position linguistic position (inclusive on both sides) in linguistic tokens should be within the specified range. Similarly to the *FTDistanceSelection*, if there are stop words in the stop-words context (see 5.7), they will be ignored, i.e. they will not be counted against the word distance.

4.10.4. Examples

We can find numbers of books containing “web”, “site”, and “usability” in their title within a window of 5 using:

```
/book[title ftcontains (“web” && “site” && “usability”) window at
most 5]/@number
```

The query

```
/book[. ftcontains (((“web” && “site”) ordered) && (“usability”
|| “testing”)) window at most 10]
```

finds the books that contain “web” and “site” in this order plus either “usability” or “testing” and all the matched linguistic tokens occur within a window of at most 10.

The query

```
/book[@number=”1”] ftsearch (“web site” && “usability”) window at
most 3
```

will return the “title” element because it contains “Web Site Usability”; the “p” element will not be returned when stop-words are not ignored because its occurrences of “web site” and “usability” are within window of 5.

4.11. *FTTimesSelection*

4.11.1. Description

FTTimesSelection controls the number of times a specified *FTSelection* condition can be matched.

4.11.2. Syntax

FTTimesSelection	::= FTSelection FTRangeSpec "occurrences"

4.11.3. Semantics

FTTimesSelection limits the number of different occurrences of *FTSelection*, which must be within the specified range.

An occurrence of the criterion is a distinct set of search token occurrences that satisfies it. For example, The *FTSelection* '("very big")' has one occurrence in the text fragment "very very big": it consists of the second "very" and "big". The *FTSelection* '"very" && "big"' has two occurrences in the text fragment "very very big": one consisting of the first "very" and "big", and the other containing the second "very" and "big". The *FTSelection* '"very" || "big"' has 7 occurrences in "very very big" – any non-empty set of words.

4.11.4. Examples

If we want to get the number of the book that contains 2 or more occurrences of "usability", we can specify:

```
/book[. ftcontains "usability" at least 2 occurrences]/@number
```

Consider the sample text fragment in 2.

The query

```
/book[@number="1" and title ftcontains ("usability" || "testing")
at most 3 occurrences]
```

will return false because "usability" occurs 3 times and "testing" occurs one time; therefore, there are 4 occurrences of "usability" || "testing".

The query

```
/book[@number="1"] ftsearch "usability" at least 2 occurrences
```

will return the "title" element because it contains 3 occurrences of "usability"; the "p" element will not be returned because it contains only one occurrence.

5. Context Modifiers

Context modifiers do not have operational semantics. They modify the operational semantics of the *FTSelection*s they are applied on.

5.1. *FTContextModifier*

5.1.1. Description

FTContextModifier production presents all possible context modifiers that set an environment for the operational semantics of *FTSelection* criteria (4.1).

5.1.2. Syntax

```
FTContextModifier      ::= FTCaseCtxMod |
                           FTDiacriticsCtxMod |
                           FTSpecialCharCtxMod |
                           FTStemCtxMod |
                           FTThesaurusCtxMod |
                           FTStopWordCtxSpec |
                           FTLanguageCtxMod |
                           FTIgnoreCtxMod
```

5.1.3. Semantics

See the semantics for each context modifier.

5.1.4. Examples

See the examples of the particular types of context specifiers.

5.2. *FTCaseCtxMod*

5.2.1. Description

This context modifier controls the way linguistic tokens are matched with regards to the letter case.

5.2.2. Syntax

```
FTCaseCtxMod        ::= "lowercase"|
                        "uppercase"|
                        "case insensitive"|
                        "case sensitive"
```

5.2.3. Semantics

FTCaseCtxMod influences the way *FTStringSelection* conditions are applied.

"`lowercase`" ("`uppercase`") specify that only linguistic tokens in lower-case (upper-case) letters can be matched exactly; "`case insensitive`" specifies that results can have both small and capital letters – their case is ignored; "`case sensitive`" specifies that the case of the letters in the result must match the case of the letters in the search token from the query.

The default is "`case insensitive`".

5.2.4. Examples

The query

```
/book[@number="1"]/title ftcontains "Usability" lowercase
```

will return false because the "title" element doen't contain "usability" (in lower case).

On the other hand,

```
/book[@number="1"]/title ftcontains "usability" case insensitive
```

will return true because the case of the letters is not considered.

5.3. *FTDiacriticsCtxMod*

5.3.1. Description

This context modifier controls the way linguistic tokens are matched with regards to the use of diacritic symbols.

5.3.2. Syntax

FTDiacriticsCtxMod	::= "`with`" "`diacritics`" |
	"`without`" "`diacritics`" |
	"`diacritics`" "`insensitive`" |
	"`diacritics`" "`sensitive`"

5.3.3. Semantics

FTDiacriticsCtxMod influences the way *FTStringSelection* conditions are applied.

"`with`" ("`without`") "`diacritics`" specifies that only linguistic tokens that contain (do not contain) diacritics can be matched exactly; "`diacritics insensitive`" specifies that there are no restrictions on the results with regards to diacritic symbols: letters containing diacritics can be matched with their non-diacritics counterparts and vice versa; "`diacritics sensitive`" specifies that the diacritic symbols must match the symbols in the search token from the query.

The default is "`diacritics insensitive`".

5.3.4. Examples

The query

```
/book[@number="1"]//editor ftcontains "Vera" with diacritics
```

will return the "editor" element, while

```
/book[@number="1"]/editors ftcontains "Véra" without diacritics
```

will return false.

5.4. *FTSpecialCharCtxMod*

5.4.1. Description

This context modifier specifies whether special characters such as punctuation should or should not be ignored.

5.4.2. Syntax

FTSpecialCharCtxMod ::= (“`with`” | “`without`”) “`special`” “`characters`”

5.4.3. Semantics

FTSpecialCharCtxMod influences the way *FTStringSelection* conditions are applied.

The modifier “`with special characters`” specifies that special characters such as punctuation must also be matched. The modifier “`without special characters`” specifies that special characters such as punctuation need not be matched.

The default is “`without special characters`”.

5.4.4. Examples

Consider the sample text fragment in section 2. The query

```
/book[@number=”1”]//editor ftcontains “Tüdor Medina” with special
characters
```

will return true, while

```
/book[@number=”1”]/editors ftcontains “Tüdor-Medina” without
special characters
```

will return false.

5.5. *FTThesaurusCtxMod*

5.5.1. Description

This context modifier controls the use of thesauri during string matching.

5.5.2. Syntax

FTThesaurusCtxMod	::= "`with`"? "`thesaurus`" Expr | "`without`" "`thesaurus`"

5.5.3. Semantics

FTThesuarusCtxSpec influences the way *FTStringSelection* conditions are applied.

The *Expr* must result in a sequence of strings (string atoms or nodes of type "xs:string") that represent valid thesauri names. Otherwise, an `error` is returned. What is a valid thesaurus name is implementation-dependent and can be either a name of system-provided or user-specified thesaurus. If *Expr* evaluates to an empty sequence, the construct is equivalent to "`without thesaurus`".

When the "`with thesaurus`" context modifier is specified, string matches also include linguistic tokens that can be found in the specified thesauri and that correspond to the query string.

The statement "`without thesaurus`" instructs the query engine not to use thesauri when matching linguistic tokens.

The default is "`without thesaurus`".

It is implementation defined how a thesaurus is represented. This includes files in a predefined format, or modules using a common interface.

5.5.4. Examples

Consider the sample text fragment in section 2. The query

```
/book[@number="1"]//p ftcontains "buttress" with thesaurus
"Synonyms"
```

will return the true if "Synonyms" is a thesaurus for synonyms in the English language.

5.6. *FTStemCtxMod*

5.6.1. Description

This context modifier controls the use of stemming during string matching.

5.6.2. Syntax

FTStemCtxMod	::= (“`with`” | “`without`”) “`stems`”

5.6.3. Semantics

FTStemCtxMod influences the way *FTStringSelection* conditions are applied.

When the ‘`with stems`’ context indicator is present, string matches may also contain linguistic tokens that have the same stem as the query string. It is implementation-defined what a stem of a linguistic is.

The clause “`without stems`” turns off the use of stemming when linguistic tokens are matched.

It is implementation-defined whether the stemming will based on an algorithm, dictionary, or mixed approach.

The default is “`without stems`”.

5.6.4. Examples

Consider the sample text fragment in section 2. The query

```
/book[@number=”1”]/title ftcontains “improve” with stems
```

will return true because it contains “improving” that has the same stem as “improve”.

5.7. *FTStopWordsCtxMod*

5.7.1. Description

This context modifier controls the use of stop words (frequent functional words such as “a”, “an”, “the”, etc. that are ignored) during string matching.

5.7.2. Syntax

FTStopWordsCtxMod ::= “`with`” “`additional`”? “`stopwords`” *Expr* ? | “`without`” “`stopwords`” *Expr?*

5.7.3. Semantics

FTStopWordsCtxMod influence the way *FTStringSelection* conditions are applied.

Expr must evaluate to a sequence of string atoms or nodes of type “xs:string”. No tokenization is performed on the strings: they are used as they occur in the sequence.

When the “`with stopwords`” context indicator is used, if a token (either in the search or linguistic) is within a collection of stop-words, it should be ignored. If *Expr* is not specified, an implementation-defined system collection of stop words is used. If *Expr* is present and “`additional`” is not specified, the strings in its result sequence are used as the new stop-word collection. If “`additional`” is specified, the strings from the result sequence are appended to the current stop-word collection. It is a syntax error to use “`additional`” without specifying an *Expr*.

“`without stopwords`” turns off the use of the linguistic tokens in the expression result as stop-words or clears the whole stop-word collection if no expression is specified.

The default is “`without stopwords`”.

5.7.4. Examples

Consider the sample text fragment in section 2. The query

```
/book[@number=”1”]//p ftcontains “usability web site” with
stopwords (“a” “the” “of”)
```

will return true. The query

```
/book[@number=”1”]//title ftcontains “usability web site” without
stopwords
```

will return false.

5.8. *FTLanguageCtxMod*

5.8.1. Description

This context modifier controls in what language the matched linguistics are considered to be.

5.8.2. Syntax

FTLanguageCtxMod ::= "`language`" *Expr*

5.8.3. Semantics

FTLanguageCtxMod influence the way *FTStringSelection* conditions are applied.

The language used can have implications in various aspect of string matching. This includes how the tokenization into linguistic tokens is performed, how are symbols transformed into lower/upper-case, what are the valid diacritic symbols, what are the possible special characters, how stemming is performed, or which linguistic tokens can considered to be stop-words. In particular, the language context may imply what are the default thesauri/stop-words sets.

Expr is an XQuery expression that must evaluate to a string atom, a node with typed value of type "xs:string", or an empty sequence.

If *Expr* evaluates to "none", "", or an empty sequence, this means that there is no language selected; otherwise, it should be valid identifier of a language.

By default, there is no language selected.

5.8.4. Examples

Consider the sample document fragment in section 2. The query

```
/book[@number="1"]//editor ftcontains "tudor" with diacritics
language "Romanian"
```

will return true.

5.9. *FTIgnoreCtxMod*

5.9.1. Description

The *FTIgnoreCtxMod* context modifier controls if the tags or the whole content of given elements should be ignored.

5.9.2. Syntax

FTIgnoreCtxMod	::= "`without`" ("`tags`" | "`content`") *Expr*

5.9.3. Semantics

The *FTIgnoreCtxMod* context modifier specifies a set of nodes the linguistic tokens in whose tag and or content should be ignored. The set of nodes is identified by the XQuery expression *Expr* that should evaluate to a sequence of element nodes.

If "`without tags`" is specified, only linguistic tokens in the start and end tag are ignored: tag name, attribute names, attribute values. If the XQuery sub-expression evaluates to an empty sequence no linguistic tokens from element tags are ignored.

If "`without content`" is specified, all the linguistic tokens in the tags and all the linguistic tokens directly contained by the elements are ignored. For example, "Web Site Usability" can be matched by "Web Usability" in the context of "`without content .//b`". If the XQuery sub-expression evaluates to an empty sequence no linguistic tokens from element tags or content are ignored.

By default no element content or tags are ignored.

5.9.4. Examples

Consider the sample document fragment in section 2. The query

```
/book[@number="1"] ftcontains "Testing" without content .//title
```

will return false because "Testing" does not occur without the "title" element whose content is ignored.

6. References

[1] Norbert Fuhr, Kai Grobjohann, “XIRQL: A Language for Information Retrieval in XML Documents”, SIGIR Conference, 2001.

[2] L. Guo, F. Shao, C. Botev, J. Shanmugasundaram, “XRANK: Ranked Keyword Search over XML Documents”, SIGMOD Conference, 2003.

[3] XQuery Language. The W3 Consortium. http://www.w3.org/TR/xquery/

[4] XQuery 1.0 and XPath 2.0 Data Model. The W3 Consortium. http://www.w3.org/TR/xpath-datamodel/

[5] XQuery and XPath Full-Text Use Cases. The W3 Consortium. http://www.w3.org/TR/xmlquery-full-text-use-cases/