MarkLogic Server

Query Performance and Tuning Guide

MarkLogic 6
September, 2012

Last Revised: 6.0-1, September, 2012

Copyright © 2012 MarkLogic Corporation. All rights reserved.

Table of Contents

Query Performance and Tuning Guide

1.0 Tuning Query Performance in MarkLogic Server 4
- 1.1 Overview of Query Performance 4
- 1.2 General Techniques to Tune Performance 5
  - 1.2.1 Search Built-In APIs 5
  - 1.2.2 Lexicons For Unique Word or Value Lookups 6
  - 1.2.3 Range Queries for Constraining Searches to a Range of Values 6
  - 1.2.4 Positions Indexes Can Help Speed Phrase Searches 6
  - 1.2.5 Use Query Meters and Query Trace to Characterize Performance 6
  - 1.2.6 Profiler API 6
  - 1.2.7 Monitoring API and Status Screens 6
  - 1.2.8 Index Options, Range Indexes, Fields 7
- 1.3 Understanding MarkLogic Server Caches 7
- 1.4 Rules of Thumb for Sizing 8

2.0 Fast Pagination and Unfiltered Searches 9
- 2.1 Understanding the Search Process 9
- 2.2 Understanding Unfiltered Searches 10
- 2.3 Using Unfiltered Searches for Fast Pagination 12
- 2.4 Example: Determining the Number of False-Positive Matches 13

3.0 Tuning Queries with query-meters and query-trace 14
- 3.1 Indexes, XPath Expressions, and Query Performance 14
- 3.2 Understanding query-meters Output 15
  - 3.2.1 Output From xdmp:query-meters 15
  - 3.2.2 Understanding the Cache Statistics 15
- 3.3 Understanding query-trace Output 17
  - 3.3.1 What query-trace Logs 17
    - 3.3.1.1 XPath Expression Analysis Messages 17
    - 3.3.1.2 Constraint Analysis Messages 18
    - 3.3.1.3 Search Execution Messages 19
  - 3.3.2 Interpreting the Log Messages 19
  - 3.3.3 Fully Searchable Paths and cts:search Operations 20
- 3.4 Using xdmp:plan to View the Evaluation Plan 21
- 3.5 Examples 21
  - 3.5.1 Sample xdmp:query-meters Output 22
  - 3.5.2 Sample xdmp:query-trace Output 23
  - 3.5.3 Logging Both query-meters and query-trace Output 23
- 3.6 General Methodology for Tuning a Query 25

4.0 Optimizing Order By Expressions With Range Indexes 26
4.1 Speed Up Order By Performance 26
4.2 Rules for Order By Optimization 26
4.3 Creating Element or Attribute (Range) Indexes 28
4.4 Example Order By Queries 28
4.4.1 Order by a Single Element 28
4.4.2 Order by Multiple Elements 30

5.0 Profiling Requests to Evaluate Performance 31
5.1 Overview of MarkLogic Server Profiling 31
5.1.1 Definitions and Terminology 31
5.1.2 MarkLogic Server Profiling Requirements Capabilities 32
5.2 Profile APIs 33
5.3 Profiling Examples 34
5.3.1 Simple Enable and Disable Example 34
5.3.2 Returning a Part of the Profile Report 36

6.0 Disk Storage Considerations 37
6.1 Disk Storage and MarkLogic Server 37
6.2 Fast Data Directory on Forests 37
6.3 Large Data Directory on Forests 38

7.0 Monitoring MarkLogic Server Performance 39
7.1 Ways to Monitor Performance and Activity 39
7.1.1 Server Logs 39
7.1.2 Status Screens in the Admin Interface 40
7.1.3 Create Your Own Server Reports 42
7.2 Server Monitoring APIs 42

8.0 Technical Support 43

Combined Product Notices 44
COPYRIGHT 44
TRADEMARK NOTICE 56

1.0 Tuning Query Performance in MarkLogic Server

This chapter describes some general issues involving query performance in MarkLogic Server, and includes the following sections:

- Overview of Query Performance
- General Techniques to Tune Performance
- Understanding MarkLogic Server Caches
- Rules of Thumb for Sizing

1.1 Overview of Query Performance

MarkLogic Server is designed to search extremely large content sets, while providing fine-grained control over the search and access of the content. Performance is always an important component in a search application. In many cases, applications will be extremely fast with no tuning whatsoever. There are, however, many tools and techniques to help make queries faster.

There are several things to consider when looking at query performance:

- Application requirements: how fast does performance need to be for your application? It is often useful to quantify this at application design time. Factors such as who will be using the application, what any user expectations for performance are, and whether the application will be publicly available are important considerations in defining performance requirements.
- Indexing options: what indexes are defined for the database? Indexing options play an important role in how well queries can be resolved from the indexes. The fastest way to resolve a query is directly from the indexes. For details on database options, see the chapters Databases and Text Indexing in the *Administrator's Guide*.
- XQuery code: is your code written in the most efficient way possible? Sometimes, code runs more slowly than necessary because there are redundant or unneeded function calls. Or there may be a MarkLogic XQuery built-in function that performs an equivalent task more efficiently. Functions such as `xdmp:estimate`, `cts:search`, lexicon functions, and so on are all designed for fast performance.
- More indexes and lexicons: can range indexes and lexicons speed up your queries? For queries that access values and/or do comparisons on those values, range indexes can greatly speed performance. Range indexes are memory mapped structures, so they can retrieve the values without ever needing to access the documents. Lexicons are lists of words or values, and they too can greatly speed up certain types of queries.

- Server tuning: are your server parameters set appropriately for your system? In most cases, the parameters set during installation work well for the system in which MarkLogic Server is installed. Nevertheless, there are cases where you might need to change some parameters, either for a short-term need or for ongoing needs.
- Scalability: is your system sufficiently large for your needs? Memory, disk space and quality, swap space, number of processors, and number of servers all contribute to the overall scalability of a MarkLogic Server system. MarkLogic Server is designed to scale to very large clusters with extremely large amounts of content.

This chapter and this book, as well as the *Application Developer's Guide*, provide information and techniques on tuning a system for optimal performance. The nature of tuning exercises is that they tend to be content-specific, so you cannot always pinpoint a particular recipe that will work for every situation. Getting to know the tools available, the XQuery APIs, and how MarkLogic Server works is the best way to make your applications run extremely fast.

1.2 General Techniques to Tune Performance

This section lists some general techniques useful in tuning performance, and provides links to places in the documentation where there is more information on a subject. It contains the following parts:

- Search Built-In APIs
- Lexicons For Unique Word or Value Lookups
- Range Queries for Constraining Searches to a Range of Values
- Positions Indexes Can Help Speed Phrase Searches
- Use Query Meters and Query Trace to Characterize Performance
- Profiler API
- Monitoring API and Status Screens
- Index Options, Range Indexes, Fields

1.2.1 Search Built-In APIs

The search built-in XQuery APIs are designed to provide very fast searches. The APIs (`cts:search`, `xdmp:estimate`, `cts:element-values`, and so on) use the indexes for fast search performance. The composable `cts:query` constructors make it easy to compose complex search queries with fast performance. For details on the search built-in XQuery APIs, see *MarkLogic XQuery and XSLT Function Reference*. For details on the constructors, see Composing cts:query Expressions in the *Search Developer's Guide*.

1.2.2 Lexicons For Unique Word or Value Lookups

MarkLogic Server allows you to create lexicons, which are lists of unique words or values in a database. Lexicons allow for very fast lookups, and in the case of values, also provide very fast counts. For details on lexicons, see the chapter Browsing With Lexicons in the *Search Developer's Guide*.

1.2.3 Range Queries for Constraining Searches to a Range of Values

Range queries allow you to specify queries that use range indexes in a `cts:query` expression. Range queries can both improve performance and make it easier to build applications that constrain on values. For details on range queries, see Using Range Queries in cts:query Expressions in the *Search Developer's Guide*.

1.2.4 Positions Indexes Can Help Speed Phrase Searches

If you specify `word positions` in the database configuration, it can speed phrase searches. During the index resolution phase of query processing, MarkLogic Server determines if words are next to each other based on their positions. For example, if you search for the phrase `"to be or not to be"`, MarkLogic Server can eliminate as possible matches, based on positions, most occurrences of these common words because they do not have the proper word next to it. This speeds performance in two ways: it lowers the number of I/Os needed to retrieve candidate fragments, and it makes the filtering phase faster because there are less candidate fragments to filter. For details about how search processing works, see "Understanding the Search Process" on page 9.

1.2.5 Use Query Meters and Query Trace to Characterize Performance

There are two XQuery functions to help you characterize the performance of queries: `xdmp:query-meters` and `xdmp:query-trace`. The former provides timings of a query and the latter logs details of the query evaluation to the `ErrorLog.txt` file. For details on these APIs, see "Tuning Queries with query-meters and query-trace" on page 14 and the *MarkLogic XQuery and XSLT Function Reference*.

1.2.6 Profiler API

MarkLogic Server has a profiler to help determine where a query is spending time processing. For details on the profiler, see "Profiling Requests to Evaluate Performance" on page 31 and the *MarkLogic XQuery and XSLT Function Reference*.

1.2.7 Monitoring API and Status Screens

There are APIs and status screens in the Admin Interface to monitor activities on your system. These can be useful in identifying bottlenecks on your system. For details, see "Monitoring MarkLogic Server Performance" on page 39.

1.2.8 Index Options, Range Indexes, Fields

There are many types of index options, including several types of wildcard indexes, element indexes, stemmed indexes, element and attribute range indexes, and so on. Depending on your needs, these indexes can help speed performance. Indexes tend to take more disk space and increase loading times, but can greatly improve performance.

Fields are another way of improving performance, especially if you are only interested in searching through certain included elements, or you want your searches to exclude particular elements. For details on fields, see Fields Database Settings in the *Administrator's Guide*.

1.3 Understanding MarkLogic Server Caches

MarkLogic Server has several caches used in query processing, defined on the group configuration page. The *list cache* stores termlists in memory, the *compressed tree cache* stores compressed fragment data in memory, and the *expanded tree cache* stores uncompressed fragment data in memory. Additionally, there are several other caches used for security objects, modules, schemas, and so on; these other caches cannot be configured. In most cases, if the caches fill up, they will move older data out to make room for newer content.

In some cases, however, it is possible to run a query that will fail because a cache was full. Particularly, when the expanded tree cache gets full, a query can fail with an `XDMP-TREECACHEFULL` exception. The following are some guidelines to avoid `XDMP-TREECACHEFULL` errors:

- Avoid queries that return the entire database. Instead, return the results in batches (a page at a time, like a classic search page, for example).
- Try to rewrite the query in a more efficient way.
- Make sure swap space is configured properly on your server.
- If you do not have sufficient memory on your server, consider adding more memory to the system.
- You can raise the sizes of the caches, but that might be a temporary fix.
- 64-bit systems are recommended. 64-bit systems can hold a lot more memory, and more memory means larger caches.

1.4 Rules of Thumb for Sizing

The following are some rule-of-thumb sizing recommendations. These recommendations are best practices based on experience with MarkLogic Server implementations. Also, some of these recommendations are content specific. Performing experiments on your own content is a good way to validate any expansions of these rules of thumb, but these provide a good starting point.

- You should have approximately 10-20 GB of forest data per 1 GB of RAM. More memory will help, too, especially if you have a lot of range indexes and/or lexicons.
- 64-bit systems are good—they greatly increase the address space so you can address more than 4GB of RAM.
- Forests should not grow too much past 200GB or 32-million fragments; when a forest size approaches 200 GB or 32-million fragments, think about creating a new forest.
- Make sure you have at least 2x memory for swap space (or the recommended amount for your platform, as described in Memory, Disk Space, and Swap Space Requirements in the *Installation Guide*). This is important to make sure MarkLogic Server does not run out of memory. At query time, MarkLogic Server asks the operating system to reserve both memory and swap space. If there is not enough of either, the query can fail with `SVC-MEMALLOC` messages. These messages can happen if you do not have the recommended amount of swap space. If you do have enough swap space and still get these errors, it can indicate that you either need to increase the amount of memory in the system or lower the amount of memory being used, either by modifying your queries or lowering some of the sizes of server caches, lowering the number of threads the server can service, and so on.
- For updates, make the journal size larger if you have a lot of range index data. A symptom of this as a problem is journal-full errors.
- For updates, make the journal size larger if your transactions span multiple forests. The journals must keep the lock information for all documents involved in the transaction, not just for the documents in the journal for the forest in which the document exists. A symptom of this as a problem is journal-full errors.
- There is a limit of 65k for the size of a string literal or a token in an XQuery program. If you need to input a string longer than 65k, use an external variable with the `xdmp:invoke` API. External variables are limited to a single node or a string, and in XCC are limited to string only. In XCC, if you need to input a node as an external variable, you must quote it as a string on input and then unquote it (`xdmp:unquote`) into a node in your XQuery function. Note that this limit is only for the size of a string literal or a token; XQuery program sizes are limited only by the cache size.

2.0 Fast Pagination and Unfiltered Searches

MarkLogic Server provides a powerful XML-enabled search capability through the `cts:search` XQuery built-in function. As part of the search capability, MarkLogic Server allows you to issue `cts:search` expressions that return results directly from the indexes, without performing the filtering necessary to ensure there are no false-positive results. This chapter describes the search process, including unfiltered searches, and includes the following sections:

- Understanding the Search Process
- Understanding Unfiltered Searches
- Using Unfiltered Searches for Fast Pagination
- Example: Determining the Number of False-Positive Matches

2.1 Understanding the Search Process

When evaluating `cts:search` expressions (and also when resolving XPath expressions within XQuery code), MarkLogic Server performs a two-step process.

1. A list of candidate fragment IDs is generated directly from the indexes, based on the index-resolvable criteria incorporated in the various parameters passed to `cts:search`. Fragment IDs are ordered according to relevance criteria. This step is called *index resolution*.

2. The candidate fragment IDs are used to load fragments from disk. Each fragment is then examined in order, using the complete criteria incorporated in the various parameters passed to `cts:search`, to determine if the fragment contains zero, one, or more than one result that matches the given `cts:search` expression. This step is called *filtering*.

The purpose of index resolution is to narrow the set of candidate fragments to as small a set as possible, without missing any. In some circumstances, the index resolution step can yield a precisely correct set of candidate fragments, rendering the filtering step redundant. In other circumstances, index resolution can reduce the set of candidate fragments somewhat, but in the candidate fragment list there are still false-positive results (that is, candidate fragments that in fact contain no matching results). In still other circumstances, the candidate fragment list can contain fragments that contain more than one matching result.

To better understand false-positive results, imagine a database configuration which has not enabled fast case-sensitive indexes (`fast case sensitive searches` on the database configuration page). This means that the full-text indexes only maintain direct lookups for words independent of their case. In this scenario, if you are searching for `"Dog"`, the indexes can only tell you what fragments contain the word `"dog"`, in any case-combination of text (for example, `"dog"`, `"DOG"`, `"Dog"`, `"doG"`, and so on). So when index resolution generates a candidate fragment list for `"Dog"`,

that list could include a fragment that has the word `"dog"` but not the word `"Dog"`. This is where filtering comes in—by loading that fragment and examining it prior to returning it as a match, MarkLogic Server is able to determine that it is not in fact a match for the specified query, and rule it out. The fragment is a false-positive result, and should not be returned to the query.

To understand how a candidate fragment can contain more than one match, consider a single-fragment document that contains multiple `<author>` elements as follows:

```
<author>Bruce Smith</author>
<author>Betty Smith</author>
<author>Gordon Blair</author>
```

Now consider the following query:

```
cts:search(//author, "Smith")
```

During index resolution, this query generates the fragment for that document as a single candidate fragment. In fact, that single document should generate two results—one for each of Bruce and Betty Smith. During the filtering step, MarkLogic Server identifies that there is more than one element in this document that matches the search requirements, and returns both of the first two `<author>` elements as results.

As you can see from these examples, the combination of index resolution and filtering combine to provide both performance and accuracy. The algorithm is designed to allow you to write complex queries, and have MarkLogic Server determine the most efficient path providing accurate results.

There are disadvantages to the algorithm, however. Sometimes, you might know better than the search engine, and through careful design of your XML and your fragmentation, you might know that filtering is simply unnecessary. In this case, filtering takes unneeded cycles. In another situation, if you want to jump deep into a result set (for example, retrieving the 1,000,000th result from a really large result set), the emphasis MarkLogic Server has on accuracy through filtering might provide an impractical constraint for your application, because filtering the first 999,999 results will take far too long. Furthermore, it might not matter to your application if, when you jump to the 1,000,000th result, you actually end up at *approximately* that result (even if in reality it is the 949,237th result).

Consequently, MarkLogic Server provides you with tools to influence the evaluation of `cts:search` expressions, indicating whether or not filtering is required.

2.2 Understanding Unfiltered Searches

An *unfiltered* search omits the filtering step, which validates whether each candidate fragment result actually meets the search criteria. Unfiltered searches, therefore, are guaranteed to be fast, while filtered searches are guaranteed to be accurate. By default, searches are filtered; you must specify the `"unfiltered"` option to `cts:search` to return an unfiltered search. Unfiltered searches have the following characteristics:

- They determine the results directly from the indexes, without filtering for validation. This makes unfiltered results most comparable to traditional search-engine style results.
- They include false-positive results. False-positive results can originate from a number of situations, including phrase searches containing 3 or more words, certain wildcard searches, punctuation-sensitive, diacritic-sensitive, and/or case-sensitive searches.
- They will be significantly affected by fragmentation policy.

The following are some useful guidelines for when to use unfilterered searches:

- You should only perform unfiltered searches on top-level nodes or on fragment roots, otherwise you might get unexpected answers. This is because, for queries below the fragment level, there is no guarantee that a particular unfiltered search even matches the query (that is, there is a match somewhere in the fragment, but not necessarily a match in the node you are searching).
- If you choose to specify an XPath other than a top-level node or a fragment root, your XPath expression will give correct results if there is only one possible node to match in each fragment (or if the only possible match is in the first node specified). This is because unfiltered searches stop after encountering the first node per fragment that matches the specified XPath expression. If you are sure that the node you specify only has one instance per fragment, then it will not miss any results (although it might get false-positive results). An example of this is `ABSTRACT` in MEDLINE citatation, where `ABSTRACT` is below the fragment root, but there is only one `ABSTRACT` node per fragment. If you specify below a fragment root and there are multiple nodes in the fragment, the search may miss results (it will only find results if they are in the first fragment).

Finally, it is useful to understand that `cts:contains` implements the filtering step of the two-step search process:

```
unfiltered cts:search + cts:contains = normal (filtered) cts:search
```

Breaking a `cts:search` operation into an unfiltered search and a `cts:contains` allows you to do the search so it is always fast, but then only do the false-positive result removal if you want or need to. This is true as long as the first parameter to `cts:search` is at a fragment root node. If it is below a root node, it is only true if you know that the first node is the only possible hit for the search (for example, if there is only one node, as in the `ABSTRACT` example above).

2.3 Using Unfiltered Searches for Fast Pagination

There are many useful applications for unfiltered searches. Applications of unfiltered searches tend to have one or more of the following characteristics:

- Your content and search terms are such that you know the unfiltered searches are also accurate (for example, the searches are all performed at document or fragment roots, they are single-term queries, and are not wildcard, punctuation-sensitive, diacritic-sensitive, and/or capitalization-sensitive searches).
- You do not mind if there are some false-positive results because the results are an estimate (that is, they need to be fast, but are not required to be exact).
- Your searches return a large number of results and you want efficient ways to jump to a particular portion of those results.

The last point describes the situation for fast pagination. *Fast pagination* is a way to get an approximate count of the total number of result hits and then provide efficient ways to jump deep to an arbitrary point in the result sequence. Such pagination is common in search engine-style results, where a particular result might return 1 million hits and the search interface returns them 10 at a time. There is usually some sort of counter that says something like “displaying 1-10 of 1,000,000 results,” and then there are links to go to the next page of results or to go to the tenth, twentieth, or any page of the results. Often it is not critical that going to the twentieth page actually gets you to the 200th hit; it is OK if there were some false-positive results, and when you click that link you actually get to the 176th result.

When you implement a fast pagination application, you will need to jump into a position in an unfiltered search. To maximize the efficiency of this search, you must immediately follow the unfiltered `cts:search` expression with the position predicate, with no XPath steps in between. For example, to jump into the 1,000,001st hit of an unfiltered search for the phrase “one two three”, the search might look like the following:

```
cts:search(fn:doc(), "one two three", "unfiltered")[1000001 to 1000010]
```

This search will skip directly to the 1,000,001st unfiltered hit and return the 10 fragments specified in the position predicate; it will not need to fetch any other fragments.

2.4 Example: Determining the Number of False-Positive Matches

The following code sample prints out the number of false-positive matches from a search.

```
xquery version "1.0-ml";
declare boundary-space preserve;
declare namespace qm="http://marklogic.com/xdmp/query-meters";

let $trueCounter := 0
let $falseCounter := 0
let $searchTerms := "one! two three"
let $x :=
  for $result in cts:search(fn:doc(), $searchTerms, "unfiltered")
  return
  (
  if ( cts:contains($result, $searchTerms) )
  then ( xdmp:set($trueCounter, $trueCounter + 1) )
  else ( xdmp:set($falseCounter, $falseCounter + 1) )
  )
return
<results>
  <resultTotal>{$trueCounter}</resultTotal>
  <false-positiveTotal>{$falseCounter}</false-positiveTotal>
  <elapsed-time>{xdmp:query-meters()/qm:elapsed-time/text()}
  </elapsed-time>
</results>
```

Because the specified `$searchTerms` contains punctuation in the middle of the phrase, any document that has the phrase “one two three” will prove to be a false-positive result. If you substitute in your query terms for the `$searchTerms` variable, you can see if your own unfiltered search yields false-positive results.

The above code uses the `xdmp:set` function to keep track of the number of matches and the number of false-positive results. It runs the unfiltered search and then uses `cts:contains` to check if each result is actually a match. If it is a match, then increment the `$trueCounter` variable, otherwise increment the `$falseCounter` variable.

3.0 Tuning Queries with query-meters and query-trace

MarkLogic Server is designed for very fast query performance over large amounts of data. While query performance is usually very fast, sometimes you will issue queries that do not perform as well as you would like. MarkLogic Server includes functions to help you optimize the performance of queries.

This chapter describes how to use the `xdmp:query-meters` and `xdmp:query-trace` functions to understand and tune the performance of queries. It includes the following sections:

- Indexes, XPath Expressions, and Query Performance
- Understanding query-meters Output
- Understanding query-trace Output
- Using xdmp:plan to View the Evaluation Plan
- Examples
- General Methodology for Tuning a Query

3.1 Indexes, XPath Expressions, and Query Performance

When you load data into a MarkLogic Server database, indexes are created based on the index configuration for that database. The indexes help to optimize searches, XPath expressions, and other query patterns.

Sometimes, however, a query cannot use the indexes, and that leads to slower performance. In these cases, there are two main types of things you can do to speed up the query performance:

- Rewrite the query so it makes better use of the indexes.
- Add more indexes.

The `xdmp:query-meters` and `xdmp:query-trace` functions provide information to help you determine where the problem areas in the query are, and can help you determine ways to easily and, in many cases, dramatically improve query performance. Understanding the output of these functions is the key to analyzing a query and tuning it for maximum performance.

To use these functions in a query:

- Add `xdmp:query-meters()` to the end of a query, with the concatenate operator (`,`) before the function.
- Add `xdmp:query-trace(true())` to the beginning of the portion of the query you want to analyze, with the concatenate operator (`,`) after the function. Then add `xdmp:query-trace(false())` at the end of the portion of the query you want to analyze, with the concatenate operator (`,`) before the function.

3.2 Understanding query-meters Output

The `xdmp:query-meters` function provides statistics about query execution. To use `xdmp:query-meters`, concatenate the `xdmp:query-meters()` function to the end of your query. For example, the following query produces both the initial query results and the `query-meters` output:

```
doc("/myDocuments/hello.xml")//a/b/c
, xdmp:query-meters()
```

The result is a sequence of `c` nodes from the `/myDocuments/hello.xml` document followed by a `qm:query-meters` node containing the `query-meters` output.

For its function signature, see the `xdmp:query-meters` function in *MarkLogic XQuery and XSLT Function Reference*.

The following subsections describe the output of the `xdmp:query-meters` function:

- Output From xdmp:query-meters
- Understanding the Cache Statistics

3.2.1 Output From xdmp:query-meters

The `xdmp:query-meters` function produces an XML document that conforms to the `query-meters.xsd` schema. The `query-meters.xsd` schema is loaded into the schemas database and is copied to the `<install_dir>/Config` directory at installation time.

The output shows elapsed time for the query, hits and misses from the various query caches, and information about fragments and documents the query accessed. The fragment output prints one element per fragment root name (not one element per fragment). The document output prints one element per document URI. For sample `xdmp:query-meters` output, see “Sample xdmp:query-meters Output” on page 22.

3.2.2 Understanding the Cache Statistics

There are several elements in the `xdmp:query-meters` output that list the number of hits and misses on the query caches. Cache hits are good, and indicate the query is running in an optimized fashion. Cache misses indicate that the query could not retrieve its results directly from the cache, and had to read the data from disk. Because disk I/O is expensive relative to reading from memory, cache misses indicate that the query might be able to be optimized, either by rewriting the parts of the query that have cache misses to better take advantage of the indexes or by adding indexes that the query can use.

MarkLogic Server has several different caches used for query processing. In general, these caches load index data into memory, providing optimized query processing for a large variety of queries.

The `xdmp:query-meters` function lists hits and misses for the following caches:

- list cache

  The *list cache* holds search term lists in memory and helps optimize XPath expressions and text searches.

- compressed tree cache

  The *compressed tree cache* holds compressed XML tree data in memory. The data is cached in memory in the same compressed format that is stored on disk.

- expanded tree cache

  The *expanded tree cache* holds the uncompressed XML data in memory (in its expanded format).

- in-memory cache

  The *in-memory cache* holds data that was recently added to the system and is still in an in-memory stand; that is, it holds data that has not yet been written to disk.

- value cache

  The *value cache* exists only for the duration of a query. It holds typed values and optimizes queries that perform frequent conversion of nodes to typed values. Each miss for the value cache indicates that an XML node must be converted to a typed value.

- regular expression cache

  The *regular expression cache* (`regexp-cache`) exists only for the duration of a query. It holds compiled regular expressions, and optimizes queries that use a regular expression multiple times.

- link cache

  The *link cache* exists only for the duration of a query. The link cache holds the relationships between parent and child nodes, reusing that relationship throughout the query execution to optimize query processing.

The cache hits and misses are also broken down by fragment and by document. Each fragment element represents all of the fragments with the specified name. Each document element represents a document with the specified URI. The fragment and document elements of the `xdmp:query-meters` output show cache hits and misses for the expanded tree cache. These statistics can help you isolate which documents or fragments are being optimally processed. If a given document or fragment gets cache misses, you might be able to add indexes or rewrite the query to speed performance.

To help tune query performance, run the `xdmp:query-meters` function with your query and look for cache misses in the `xdmp:query-meters` output; cache misses indicate areas where the query can be tuned (either by rewriting or by adding indexes) for better performance.

3.3 Understanding query-trace Output

The `xdmp:query-trace` function logs output to the `<data_dir>/Logs/ErrorLog.txt` file during query execution. To start query tracing, concatenate the `xdmp:query-trace(true())` function at the part of your query where you want the tracing to begin, and add `xdmp:query-trace(false())` where you want tracing to stop. For example, the following query produces results for the query and logs the `query-trace` output to the `ErrorLog.txt` file:

```
xdmp:query-trace(true()),
doc("/myDocuments/hello.xml")//a/b/c
, xdmp:query-trace(false())
```

For its function signature, see the `xdmpquery-trace` function in *MarkLogic XQuery and XSLT Function Reference*.

The following subsections describe the output of the `xdmp:query-trace` function:

- What query-trace Logs
- Interpreting the Log Messages
- Fully Searchable Paths and cts:search Operations

3.3.1 What query-trace Logs

The `xdmp:query-trace` function prints INFO-level messages to the log file while a query is executing. It prints one log message for each XPath expression, and at least one log message for each step in the XPath expression. It also prints messages for predicates and other parts of query evaluation. Therefore, `xdmp:query-trace` can potentially log a large number of messages to the log file, particularly for complex queries that contain very deep XPath expressions and many searches.

The `xdmp:query-trace` function logs the following information about the query processing and execution:

- XPath Expression Analysis Messages
- Constraint Analysis Messages
- Search Execution Messages

3.3.1.1 XPath Expression Analysis Messages

The `xdmp:query-trace` function prints INFO-level messages to the log file about the XPath expressions in the query. The messages log whether an XPath expression is `searchable`. A `searchable` expression is one which can be optimized by using the indexes. The `query-trace` output shows which steps in the XPath expression are or are not `searchable` with the indexes.

For query tuning, the most important thing the log output has is the information about whether an expression is `searchable` or not. In general, `searchable` expressions can use the indexes to execute, and therefore execute fast. Expressions that are `unsearchable` cannot use the indexes, and must fetch the data from disks. For a summary of how to read the log messages, see "Interpreting the Log Messages" on page 19.

3.3.1.2 Constraint Analysis Messages

The constraint analysis phase of the `query-trace` output prints log messages about predicates in XPath expressions and `where` clauses. At the beginning of each constraint analysis section, you will see a message similar to the following:

```
2004-12-06 11:57:18.325 Info: line 21: Gathering constraints.
```

The output logs one message for each step in the XPath expression that contributes to the constraint. It only prints messages about constraints that can be evaluated using the indexes; unoptimized constraints do not generate any `query-trace` output. When the predicate constraint is reached, the log shows a message similar to the following:

```
2004-12-15 10:44:57.734 Info: line 2: Comparison contributed hash value
constraint: Heading-2 = "hello"
```

This message corresponds to an XPath expression with a predicate like the following:

```
doc("/myDocuments/hello.xml")/XML//Heading-2[. = "hello"]
```

The log message text `hash value constraint` indicates that the optimizer used the standard indexes (word search, stemmed search, and so on, as set up in the database configuration) to evaluate this predicate. Equality constraints on predicates use the standard indexes for evaluation, and this makes the evaluation perform fast.

Inequality constraints such as greater than (`gt` or `>`) and less than (`lt` or `<`) cannot be evaluated using the standard indexes. For inequality constraints to be optimized, you must have an element (range) index on the element used in the comparison. If you have an inequality constraint and have an element index on the element used in the comparison, the log shows a message similar to the following for the constraint `Heading-2 > "hello"`:

```
2004-12-15 10:44:57.734 Info: line 2: Comparison contributed range
value constraint: Heading-2 > "hello"
```

The log message text `range value constraint` indicates that the optimizer used an element index to evaluate the query.

> **Note:** If neither the standard indexes nor an element index is used to evaluate a constraint, no such log message appears, and the constraint is not optimized.

3.3.1.3 Search Execution Messages

The `xdmp:query-trace` function also logs detailed information about how many fragments are used to evaluate a query. These messages show the number of fragments that are filtered. When a fragment is filtered, it means that the indexes found a possible match for the query in that fragment, and the fragment must then be retrieved to make sure it meets all of the query criteria. In a well-optimized query, the number of fragments filtered will be close to the number of fragments that satisfy the query.

If a query returns no results, or if it can be answered directly from the indexes, there will be no fragments filtered, and the log shows messages similar to the following:

```
2004-12-15 10:44:57.367 Info: line 2: Executing search.
2004-12-15 10:44:57.367 Info: line 2: Selected 0 fragments to filter
```

If the query results come from a single fragment, and the query uses either the standard or element (range) indexes for its evaluation, the log shows messages similar to the following:

```
2004-12-15 11:14:10.926 Info: line 2: Executing search.
2004-12-15 11:14:10.926 Info: line 2: Selected 1 fragment to filter
```

The line that says `Selected 1 fragment to filter` indicates how many candidate fragment references were returned from the index resolution stage of query processing. For a query that makes good use of the indexes, the number of fragments filtered is close to the number of fragments returned in the query results. For example, if there are 45 fragments that match a given query, and if `xdmp:query-trace` shows 45 fragments filtered, then that query is making good use of the indexes (because it does not have to filter any fragments that end up not contributing to the query result).

In most cases, the smaller the number of fragments selected to filter, the faster the query performs. An exception to this is if you are doing unfiltered searches, as unfiltered search skip the filtering stage of query processing. For details on unfiltered searches, see “Fast Pagination and Unfiltered Searches” on page 9.

3.3.2 Interpreting the Log Messages

The messages written to the log from the `xdmp:query-trace` function help you to determine if there are ways to optimize the performance of a query. The following is a summary of some important things to look for when interpreting the `xdmp:query-trace` output:

- The output is written to the `ErrorLog.txt` file.
- Log messages with the term `searchable` are good—this means indexes can be used to execute this part of the query (which in turn means the query will execute fast).
- Suspect problem areas when you see log messages with the term `unsearchable`—this means the indexes cannot be used to execute this part of the query.

- Log messages with the term `does not use indexes` mean that there might be XPath steps below this step that are `searchable`, but this step or predicate will not be resolved directly from the indexes (known as *conditionally searchable*). This is not necessarily bad, as searches with steps that do not use the indexes can still be fast, but it is not as good as `searchable`.
- Log messages with the term `comparison contributed` **`hash`** `value` constraint indicate that this predicate used the standard indexes to execute (which in turn indicates an optimized predicate evaluation).
- Log messages with the term `comparison contributed` **`range`** `value` constraint indicate that this predicate used an element (range) index to execute (which in turn indicates an optimized predicate evaluation).
- No `hash` or `range` message in the constraint section indicates that the constraint needed to scan the fragment to execute, and could not be optimized from the indexes.
- In the execution phase, the `xdmp:query-trace` output has a log message indicating the number of fragments filtered. In a fully optimized query, that number is equal to the number of fragments that the query returns (the number you would get if you wrapped the search portion of the query in an `xdmp:estimate` call). As the number of fragments filtered increases, and particularly as the number of fragments filtered grows past the number of fragments that ultimately match the query, the amount of work needed to execute the query increases (which in turn causes performance to slow).
- XPath predicates that cross fragment boundaries are `unsearchable` (cannot use indexes). For example, if a document is fragmented at the `b` element, then you should make sure predicates do not cross the b boundary. Therefore, the following expression:

  ```
  /a/b[c="1"]/../d
  ```

  will run faster than the following expression:

  ```
  /a[b/c="1"]/d
  ```

3.3.3 Fully Searchable Paths and cts:search Operations

Queries that use the built-in search operation `cts:search` require that the XPath expression searched is *fully searchable*. A fully searchable path is one that has no steps that are `unsearchable` and whose last step is `searchable`. Some steps of the XPath expression might not use an index directly (that is, they are *conditionally searchable*), but as long as *no* steps are `unsearchable` and the *last* step is `searchable`, it is said to be fully searchable. You can use the `xdmp:query-trace` function to see if the path is fully searchable. If there are no entries in the `xdmp:query-trace` output indicating that a step is `unsearchable`, then that path is fully searchable. Queries that use `cts:search` on `unsearchable` XPath expressions will fail with an an error message. You can often make the path expressions fully searchable by rewriting the query or adding new indexes.

A *partially searchable* XPath expression is one whose first step is searchable, but does not otherwise meet the requirements to be fully searchable. Partially searchable expressions will use the indexes for XPath evaluation, but will not be allowed as the first parameter to `cts:search`.

XPath expressions must be fully searchable for optimizing `order by` expressions, too. For details on optimizing `order by` expressions, see "Optimizing Order By Expressions With Range Indexes" on page 26.

3.4 Using xdmp:plan to View the Evaluation Plan

You can use the `xdmp:plan` built-in function to see the search and execution plan for a query. It takes an XQuery expression, and it it returns an XML report providing information about how the indexes will be used if you were to run the expression. It provides much of the information shown in `xdmp:query-trace`, as well as some more information about the query terms selected from the index. The `xdmp:plan` output is useful in determining if an expression is optimized properly and if your range indexes are being used as you expect them to be.

Running an `xdmp:plan` on a search is similar to running an `xdmp:estimate` on a search, and the results of the estimate are included in the `xdmp:plan` output. If the search cannot be run in a plan or estimate, then it will throw an `XDMP-UNSEARCHABLE` exception. For more details and the signature of `xdmp:plan`, see the *MarkLogic XQuery and XSLT Function Reference*.

3.5 Examples

This section shows sample output from the `xdmp:query-meters` and `xdmp:query-trace` functions. The following examples are included:

- Sample xdmp:query-meters Output
- Sample xdmp:query-trace Output
- Logging Both query-meters and query-trace Output

3.5.1 Sample xdmp:query-meters Output

The following listing shows sample output from the `xdmp:query-meters` function:

```
<qm:query-meters xsi:schemaLocation="http://marklogic.com/xdmp/query-m
eters query-meters.xsd"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:qm="http://marklogic.com/xdmp/query-meters">
  <qm:elapsed-time>PT0S</qm:elapsed-time>
  <qm:list-cache-hits>5</qm:list-cache-hits>
  <qm:list-cache-misses>0</qm:list-cache-misses>
  <qm:in-memory-list-hits>0</qm:in-memory-list-hits>
  <qm:expanded-tree-cache-hits>1</qm:expanded-tree-cache-hits>
  <qm:expanded-tree-cache-misses>0</qm:expanded-tree-cache-misses>
  <qm:compressed-tree-cache-hits>0</qm:compressed-tree-cache-hits>
  <qm:compressed-tree-cache-misses>0</qm:compressed-tree-cache-misses>
  <qm:in-memory-compressed-tree-hits>0
       </qm:in-memory-compressed-tree-hits>
  <qm:value-cache-hits>0</qm:value-cache-hits>
  <qm:value-cache-misses>0</qm:value-cache-misses>
  <qm:regexp-cache-hits>0</qm:regexp-cache-hits>
  <qm:regexp-cache-misses>0</qm:regexp-cache-misses>
  <qm:link-cache-hits>0</qm:link-cache-hits>
  <qm:link-cache-misses>0</qm:link-cache-misses>
  <qm:fragments-added>0</qm:fragments-added>
  <qm:fragments-deleted>0</qm:fragments-deleted>
  <qm:fragments>
    <qm:fragment>
      <qm:root xmlns="">root_name</qm:root>
      <qm:expanded-tree-cache-hits>1</qm:expanded-tree-cache-hits>
      <qm:expanded-tree-cache-misses>0</qm:expanded-tree-cache-misses>
    </qm:fragment>
  </qm:fragments>
  <qm:documents>
    <qm:document>
      <qm:uri>/myDocuments/hello.xml</qm:uri>
      <qm:expanded-tree-cache-hits>1</qm:expanded-tree-cache-hits>
      <qm:expanded-tree-cache-misses>0</qm:expanded-tree-cache-misses>
    </qm:document>
  </qm:documents>
</qm:query-meters>
```

3.5.2 Sample xdmp:query-trace Output

The following sample query:

```
xdmp:query-trace(true()),
doc("/myDocs/file.xml")//Node-2
```

produces the following `xdmp:query-trace` output in the `ErrorLog.txt` file:

```
2004-12-08 15:27:27.926 Info: line 2: Analyzing path:
     doc("/myDocs/file.xml")/descendant::Node-2
2004-12-08 15:27:27.926 Info: line 2: Step 1 is searchable:
     doc("/myDocs/file.xml")
2004-12-08 15:27:27.926 Info: line 2: Step 2 axis does not use
     indexes: descendant
2004-12-08 15:27:27.926 Info: line 2: Step 2 test is searchable: Node-2
2004-12-08 15:27:27.926 Info: line 2: Step 2 is searchable:
     descendant::Node-2
2004-12-08 15:27:27.926 Info: line 2: Path is searchable.
2004-12-08 15:27:27.926 Info: line 2: Gathering constraints.
2004-12-08 15:27:27.926 Info: line 2: Step 2 test contributed 1
     constraint: Node-2
2004-12-08 15:27:27.926 Info: line 2: Executing search.
2004-12-08 15:27:27.926 Info: line 2: Selected 1 fragment to filter
```

3.5.3 Logging Both query-meters and query-trace Output

You can use the `xdmp:log` function to write the `xdmp:query-meters` output to the log file with the `xdmp:query-trace` output as follows:

```
xdmp:log("
****
**** Begin query trace and meter log
****
"),
xdmp:query-trace(true()),
doc("/myDocs/file.xml")//Heading-2[. = "hello"]
,
xdmp:log(xdmp:query-meters())
,
xdmp:log("
****
**** End query trace and meter log
****
")
```

This query produces log output in the `ErrorLog.txt` file like the following:

```
2004-12-08 15:48:01.502 Info:

****
**** Begin query trace and meter log
```

```
****

004-12-08 15:48:01.502 Info: line 9: Analyzing path:
     doc("/myDocs/file.xml")/descendant::Node-1
2004-12-08 15:48:01.502 Info: line 9: Step 1 is searchable:
     doc("/myDocs/file.xml")
004-12-08 15:48:01.502 Info: line 2: Step 2 axis does not use
     indexes: descendant
004-12-08 15:48:01.502 Info: line 2: Step 2 test is searchable: Node-2
004-12-08 15:48:01.502 Info: line 2: Step 2 is searchable:
     descendant::Node-2
004-12-08 15:48:01.502 Info: line 2: Path is searchable.
004-12-08 15:48:01.502 Info: line 2: Gathering constraints.
2004-12-08 15:48:01.502 Info: line 2: Step 2 test contributed 1
     constraint: Node-2
2004-12-08 15:48:01.502 Info: line 2: Executing search.
004-12-08 15:48:01.502 Info: line 2: Selected 1 fragment to filter
2004-12-08 15:48:01.502 Info: <qm:query-meters
xsi:schemaLocation="http://marklogic.com/xdmp/query-meters
query-meters.xsd"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:qm="http://marklogic.com/xdmp/query-meters">
  <qm:elapsed-time>PT0.01S</qm:elapsed-time>
  <qm:list-cache-hits>4</qm:list-cache-hits>
  <qm:list-cache-misses>0</qm:list-cache-misses>
  <qm:in-memory-list-hits>0</qm:in-memory-list-hits>
  <qm:expanded-tree-cache-hits>0</qm:expanded-tree-cache-hits>
  <qm:expanded-tree-cache-misses>0</qm:expanded-tree-cache-misses>
  <qm:compressed-tree-cache-hits>0</qm:compressed-tree-cache-hits>
  <qm:compressed-tree-cache-misses>0</qm:compressed-tree-cache-misses>
  <qm:in-memory-compressed-tree-hits>0</qm:in-memory-compressed-tree-h
its>
  <qm:value-cache-hits>0</qm:value-cache-hits>
  <qm:value-cache-misses>0</qm:value-cache-misses>
  <qm:regexp-cache-hits>0</qm:regexp-cache-hits>
  <qm:regexp-cache-misses>0</qm:regexp-cache-misses>
  <qm:link-cache-hits>0</qm:link-cache-hits>
  <qm:link-cache-misses>0</qm:link-cache-misses>
  <qm:fragments-added>0</qm:fragments-added>
  <qm:fragments-deleted>0</qm:fragments-deleted>
  <qm:fragments/>
  <qm:documents/>
</qm:query-meters>
2004-12-08 15:48:01.502 Info:

****
**** End query trace and meter log
****
```

3.6 General Methodology for Tuning a Query

The following are general steps you can take to analyze and tune query performance. These steps represent a methodology; the actual steps you take will depend on your application and queries.

1. Identify the application where you see query performance slower than you expect.
2. In the application, break apart different parts of the query into separate queries and run them separately.
3. If you identify code that appears to run slowly, append `xdmp:query-meters()` to the end of the code and run it again. For details, see "Understanding query-meters Output" on page 15.
4. In the `xdmp:query-meters` output, record the elapsed time and look for cache misses.
5. Run the query several times and compare the `xdmp:query-meters` output between the different runs. There are some query caches that are populated when a query runs the first time, and can improve the performance of subsequent query runs.
6. Continue to try and simplify the query, helping to isolate where it might be running slow.
7. When you have isolated the query down to as simple a case as possible, add `xdmp:query-trace(true())` to the beginning of the query and run it again. For details, see "Understanding query-trace Output" on page 17.
8. Examine the `query-trace` output in the `ErrorLog.txt` file. Look for XPath steps that are `unsearchable`.
9. If you find `unsearchable` steps, see if there are ways to rewrite the query so those steps become `searchable`.
10. Examine the `constraints` entries of the `query-trace` log output. For details, see "Constraint Analysis Messages" on page 18.
11. Check the `query-trace` log output for the number of fragments used to filter. This number should be the close to or the same as the number of fragments that match the searchable expression (the number returned from `xdmp:estimate`) if the query is well optimized.
12. Check your indexing options. Add indexes if the proper indexes are not built. For example, if stemmed or word indexes are not built, many XPath steps will be `unsearchable`. Also, if your query contains inequality constraints, you will need element (range) indexes to optimize those constraints.
13. After making query and/or index changes, rerun the query with `xdmp:query-meters()` to see if the execution time has decreased and the number of cache misses has decreased.
14. Continue iteratively with this methodology until you are satisfied that the query execution is fast and well optimized.

4.0 Optimizing Order By Expressions With Range Indexes

When you have queries that include an `order by` expression, you can create element or attribute (range) indexes on the element(s) or attribute(s) in the `order by` expression to speed performance of those types of queries. This chapter describes this optimization and how to use it in your queries, and includes the following sections:

- Speed Up Order By Performance
- Rules for Order By Optimization
- Creating Element or Attribute (Range) Indexes
- Example Order By Queries

4.1 Speed Up Order By Performance

MarkLogic Server allows you to create indexes on elements to speed up performance of queries that order the results based on that element. The `order by` clause is the "O" in the XQuery `FLWOR` expression, and it allows you to sort the results of a query based on one ore more elements. The `order by` optimization speeds up queries that order the results and then return a subset of those results (for example, the first 10 results).

4.2 Rules for Order By Optimization

The following rules apply to a query in order for the `order by` optimization to apply:

- Optimizes subsets of `order by` queries. For example:

  ```
  (FLWOR)[1 to 20]
  ```

  where `FLWOR` is an XQuery `FLWOR` expression.

- Uses element or attribute (range) indexes.
- The sequence bound to the `for` variable must be fully searchable XPath expression or a `cts:search` expression. See "Fully Searchable Paths and cts:search Operations" on page 20.
- The order by expression must be on variables bound in the `for` clause; queries that have `order by` expressions on variables bound to a sequence in a `let` clause are *not* optimized.
- There must be a range index on the last step of the order by expression. For example:

  ```
  order by $x/bar/foo
  ```

  needs a range index on `foo` to execute with the `order by` optimization.

- The type of the order by expression must be the type of the range index, either implicitly, through a schema, or through an explicit cast. If there is a cast in the order by expression, than it must be to the type of the range index.
- You can have order by expressions with multiple items, as long as there is a range index on each item. For example:

  ```
  order by $x/foo, $x/bar
  ```

  as long as there are range indexes for `foo` and `bar`.

- The XPath expression in the order by expression must be a simple relative path; no math or other expressions are allowed.
- It does not matter what the `let`, `where`, or `return` clauses are; these do not effect the optimization.
- If you order by `cts:score($x)`, `cts:confidence($x)`, or `cts:quality($x)`, no range index is required.
- You can specify either `ascending` or `descending` orders (optionally).
- You can specify either `empty greatest` or `empty least` (optionally, must be the last part of the `order by` clause).
- Optimized `order by` clauses implicitly add `order by` expressions for `cts:score` and document order to the end of the `order by` expression.
- If you have a function that is a `FLWOR` expression (with the required fully searchable path, etc.), subsets of that will be optimized. for example

  ```
  xquery version "1.0-ml";
  declare function local:foo()
  {
  for $x in //a/b/c
  order by $x/d
  return $x
  };
  ( local:foo() )[1 to 10]
  ```

- The search or XPath expression must be part of the `FLWOR`, not bound to a variable that is referenced in the `FLWOR`. For example, the following will not be optimized:

  ```
  let $x := cts:search(/foo, "hello")
  return
  (for $y in $x
  order by $y/element
  return $y)[1 to 10]
  ```

  but the following will (given the other rules are followed):

```
(for $y in cts:search(/foo, "hello")
order by $y/element
return $y)[1 to 10]
```

- You can use `xdmp:query-trace` to determine if a query is using the range indexes to optimize an order by expression. For details on using `xdmp:query-trace`, see "Understanding query-trace Output" on page 17.
- When using an order by with a `cts:search`, further optimization occurs if you specify the `"unfiltered"` option to `cts:search`. For example, if you order by a simple XPath expression and that expression returns a sequence, if the `cts:search` is `"filtered"` (which is the default) then the search will throw an exception (because it is illegal to order by a sequence of more than one item), but if you use the `"unfiltered"` option to `cts:search`, the search will complete and will use the range index. If there are multiple values that match the order by expression in an unfiltered `cts:search`, then it will use the maximum value (`fn:max($result/item())`) for `order by ascending` and the minimum value (`fn:min($result/item())`) for `order by decending`. For more details about unfiltered `cts:search`, see "Fast Pagination and Unfiltered Searches" on page 9.

4.3 Creating Element or Attribute (Range) Indexes

You must create range indexes on the elements or attributes in which you order your result by in the `order by` expression. You create range indexes using the Admin interface by going to the Databases > *database_name* > Element Indexes or Attribute Indexes page. Be sure to select the proper type for the element or attribute. For more details on creating indexes, see the *Administrator's Guide*.

4.4 Example Order By Queries

This section shows the following simple queries that use the order by optimizations:

- Order by a Single Element
- Order by Multiple Elements

4.4.1 Order by a Single Element

The following query returns the first 100 `lastname` elements. In order for this query to run optimized, there must be a range index defined on the `lastname` element.

```
(for $x in //myNode
order by $x/lastname
return
$x/lastname)[1 to 100]
```

If you enabled query tracing on this query (by adding `xdmp:query-trace(fn:true()),` to the beginning of the query, for example), the query trace output will show if the range index is being used for the optiomization. If the range index is not being used, the query-trace output looks similar to the following:

```
2009-05-15 15:56:05.046 Info: myAppServer: line 2:
xdmp:eval("xdmp:query-trace(fn:true()),&#13;&#10;(for $x in
//myNode&#13;&#...", (), <options
xmlns="xdmp:eval"><database>661882637959476934</database><modules>0</m
odules><defa...</options>)
2009-05-15 15:56:05.068 Info: myAppServer: line 2: Analyzing path for
$x: collection()/descendant::myNode
2009-05-15 15:56:05.068 Info: myAppServer: line 2: Step 1 is
searchable: collection()
2009-05-15 15:56:05.068 Info: myAppServer: line 2: Step 2 is
searchable: descendant::myNode
2009-05-15 15:56:05.068 Info: myAppServer: line 2: Path is fully
searchable.
2009-05-15 15:56:05.068 Info: myAppServer: line 2: Gathering
constraints.
2009-05-15 15:56:05.068 Info: myAppServer: line 2: Step 2 test
contributed 1 constraint: myNode
2009-05-15 15:56:05.068 Info: myAppServer: line 2: Executing search.
2009-05-15 15:56:05.089 Info: myAppServer: line 2: Selected 6 fragments
to filter.
```

The above output does not show that the range index is being used. This could be because the range index does not exist or it could indicate that one of the criteria for the order by optimizations is not met, as described in “Rules for Order By Optimization” on page 26.

When the correct range index is in place and the query is being optimized, the query-trace output will look similar to the following:

```
2009-05-15 15:58:04.145 Info: myAppServer: line 2:
xdmp:eval("xdmp:query-trace(fn:true()),&#13;&#10;(for $x in
//myNode&#13;&#...", (), <options
xmlns="xdmp:eval"><database>661882637959476934</database><modules>0</m
odules><defa...</options>)
2009-05-15 15:58:04.145 Info: myAppServer: line 2: Analyzing path for
$x: collection()/descendant::myNode
2009-05-15 15:58:04.145 Info: myAppServer: line 2: Step 1 is
searchable: collection()
2009-05-15 15:58:04.145 Info: myAppServer: line 2: Step 2 is
searchable: descendant::myNode
2009-05-15 15:58:04.145 Info: myAppServer: line 2: Path is fully
searchable.
2009-05-15 15:58:04.146 Info: myAppServer: line 2: Gathering
constraints.
2009-05-15 15:58:04.146 Info: myAppServer: line 2: Step 2 test
contributed 1 constraint: myNode
2009-05-15 15:58:04.146 Info: myAppServer: line 2: Order by clause
contributed 1 range ordering constraint for $x: order by $x/lastname
ascending
2009-05-15 15:58:04.146 Info: myAppServer: line 2: Executing search.
2009-05-15 15:58:04.183 Info: myAppServer: line 2: Selected 6 fragments
to filter.
```

Notice the line that says `Order by clause contributed 1 range constraint`. That line indicates that the query is being optimized by the range index (which is good).

4.4.2 Order by Multiple Elements

The following query returns the first 100 `myNode` elements, ordered by `lastname` and then `firstname`. For this query to run optimized, there must be a range index defined on the `lastname` and `firstname` elements.

```
(for $x in //myNode
order by $x/lastname, $x/firstname
return
$x)[1 to 100]
```

If you run query-trace with this query, that will verify whether the range indexes are being used.

5.0 Profiling Requests to Evaluate Performance

This chapter describes how to use the Performance Profiler built-in functions to examine the evaluation characteristics of XQuery requests in MarkLogic Server, and includes the following sections:

- Overview of MarkLogic Server Profiling
- Profile APIs
- Profiling Examples

5.1 Overview of MarkLogic Server Profiling

MarkLogic Server accepts XQuery requests on HTTP and XDBC servers, and those requests evaluate the XQuery program on the fly. To help understand the performance characteristics of these requests, MarkLogic Server includes an API to gather statistics about XQuery program evaluation. The statistics include time spent in various parts of the program and counts of how many times various expressions are called. The result of all of these statistics allow you to build a *profile* of the performance characteristics. This section provides an overview of the query profile capabilities in MarkLogic Server and includes the following subsections:

- Definitions and Terminology
- MarkLogic Server Profiling Requirements Capabilities

5.1.1 Definitions and Terminology

The following table lists some terms and their definitions used in describing the profile API.

Term	Definition
XQuery Program	The XQuery main module fully expanded with any XQuery library modules needed for its evaluation. An XQuery program is sometimes referred to as a query, a statement, or a request. For more details on this terminology, see Understanding Transactions in MarkLogic Server in the *Application Developer's Guide*.
profiler	An application which measures the performance characteristics of a running program (in the case MarkLogic Server, of an XQuery program).
expression	The basic parse element of an XQuery program. Expressions can represent literal values, arithmetic operations, functions, function calls, and so on. Expressions can contain other expressions.

Term	Definition
shallow time	The time spent evaluating a specific expression, not including time spent evaluating any expressions contained within the specific expression.
deep time	The total time spent evaluating an expression, including time spent evaluating any expressions contained within the specific expression.
elapsed time	Both shallow and deep time are expressed in elapsed wall clock time.
profile report	An XML report containing statistics for all of the expressions evaluated while profiling was enabled. For a sample profile report, see "Simple Enable and Disable Example" on page 34.

5.1.2 MarkLogic Server Profiling Requirements Capabilities

The profile API in MarkLogic Server is not in itself a query profile application (a *profiler*), but you could use the profile API to build such a profiler application. Such an application would measure the performance characteristics of an XQuery program. You can also simply use the profile API to generate profile reports and then either manually analyze the reports or use XQuery to extract details and format the reports.

To use the profile API, you must first enable profiling on the App Server in which your XQuery program is serviced (or on the task server if you are profiling spawned queries). The profile allow option must be set to true.



Once profiling is enabled, you can use the `prof:enable`, `prof:report`, and other profile APIs to generate profiling statistics for the evaluation of individual XQuery programs.

Profiling helps you to see where a query is spending its processing time. The statistics are gathered for activity in the evaluation portion of the query, at the individual expression level. Time spent in the data node portion of the query (time spent gathering content from the forests) is included in the expression time for the expression that requested the content from the forest (for example, a `cts:search`). For each expression, the profile report shows the shallow time and the deep time (see "Definitions and Terminology" on page 31 for these definitions).

All profiling information is gathered on a per-request basis; there is no notion of profiling a set of requests, although it is possible to write an application that performs that kind of aggregation.

5.2 Profile APIs

The following functions are included in the Performance Profiler APIs:

- `prof:allowed`
- `prof:disable`
- `prof:enable`
- `prof:eval`
- `prof:invoke`
- `prof:report`
- `prof:reset`

For details on the APIs and for their function signatures, see the *MarkLogic XQuery and XSLT Function Reference*. Note the following about the profile APIs.

- If profiling is not enabled on the App Server and in the XQuery program (via `prof:enable`), then profile APIs do not do anything except return the empty sequence.
- You can profile the currently running request (`prof:enable(xdmp:request())`), an evaluated request (`prof:eval`), or an invoked module (`prof:invoke`). To profile other requests, you need to debug the request, and debugging is not supported in MarkLogic Server 3.2.; if you try to profile another request, MarkLogic Server throws the `DBG-NOTSTOPPED` exception.
- Constants (for example, `47` or `"hello"`) do not show up in the profile report.
- Constructed elements do not show up in the profile report.
- Profile time starts after the static analysis phase of query evaluation; it does not include the query parsing time.
- All of the profile APIs are in the `http://marklogic.com/xdmp/profile` namespace. The `prof` prefix is bound to this namespace, and is pre-configured in MarkLogic Server (so there is no need to define this namespace in your XQuery prolog).
- If you profile a request besides the currently running request, or if you start a new request for profiling using `prof:eval` or `prof:invoke`, you need one of the privileges `http://marklogic.com/xdmp/privileges/profile-my-requests` (to profile a request issued by the same user ID) or `http://marklogic.com/xdmp/privileges/profile-any-requests` (to profile a request issued by any user ID). If you are profiling the currently running request, no privileges are required.

5.3 Profiling Examples

The following are code examples showing some simple usage patterns for the profile API. For details on the APIs, see the *MarkLogic XQuery and XSLT Function Reference*. This section shows the following examples:

- Simple Enable and Disable Example
- Returning a Part of the Profile Report

5.3.1 Simple Enable and Disable Example

The following examples show simple uses for the profile API.

```
let $req := xdmp:request()
let $dummy1 := prof:enable($req)
let $version := xdmp:version()
let $node :=
  <foo>
    <version>{$version}</version>
    <request>{$req}</request>
  </foo>
let $dummy2 := prof:disable($req)
let $dummy3 := fn:current-dateTime()
let $dummy4 := prof:enable($req)
let $dummy5 := 47
let $dummy6 := $node/foo/request
let $dummy6 := prof:disable($req)

return
prof:report($req)
```

This query returns the following report:

```
<prof:report
  xsi:schemaLocation="http://marklogic.com/xdmp/profile profile.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:prof="http://marklogic.com/xdmp/profile">
  <prof:metadata>
    <prof:overall-elapsed>PT0S</prof:overall-elapsed>
    <prof:created>2007-03-19T14:20:18.763-07:00</prof:created>
    <prof:server-version>3.2-20070319</prof:server-version>
  </prof:metadata>
  <prof:histogram>
    <prof:expression>
      <prof:expr-id>6467258264555963988</prof:expr-id>
      <prof:expr-source>xdmp:version()</prof:expr-source>
      <prof:uri/>
      <prof:line>3</prof:line>
      <prof:count>1</prof:count>
      <prof:shallow-time>PT0S</prof:shallow-time>
```

```
      <prof:deep-time>PT0S</prof:deep-time>
    </prof:expression>
    <prof:expression>
      <prof:expr-id>127913224145561857</prof:expr-id>
      <prof:expr-source>prof:disable($req)</prof:expr-source>
      <prof:uri/>
      <prof:line>9</prof:line>
      <prof:count>1</prof:count>
      <prof:shallow-time>PT0S</prof:shallow-time>
      <prof:deep-time>PT0S</prof:deep-time>
    </prof:expression>
    <prof:expression>
      <prof:expr-id>8377097069965042614</prof:expr-id>
      <prof:expr-source>prof:disable($req)</prof:expr-source>
      <prof:uri/>
      <prof:line>14</prof:line>
      <prof:count>1</prof:count>
      <prof:shallow-time>PT0S</prof:shallow-time>
      <prof:deep-time>PT0S</prof:deep-time>
    </prof:expression>
    <prof:expression>
      <prof:expr-id>10959125668150171452</prof:expr-id>
      <prof:expr-source>prof:enable($req)</prof:expr-source>
      <prof:uri/>
      <prof:line>2</prof:line>
      <prof:count>1</prof:count>
      <prof:shallow-time>PT0S</prof:shallow-time>
      <prof:deep-time>PT0S</prof:deep-time>
    </prof:expression>
    <prof:expression>
      <prof:expr-id>1823871875827665493</prof:expr-id>
      <prof:expr-source>$node/child::foo/child::request
        </prof:expr-source>
      <prof:uri/>
      <prof:line>14</prof:line>
      <prof:count>1</prof:count>
      <prof:shallow-time>PT0S</prof:shallow-time>
      <prof:deep-time>PT0S</prof:deep-time>
    </prof:expression>
    <prof:expression>
      <prof:expr-id>16669888445989754369</prof:expr-id>
      <prof:expr-source>prof:enable($req)</prof:expr-source>
      <prof:uri/>
      <prof:line>11</prof:line>
      <prof:count>1</prof:count>
      <prof:shallow-time>PT0S</prof:shallow-time>
      <prof:deep-time>PT0S</prof:deep-time>
    </prof:expression>
  </prof:histogram>
</prof:report>
```

You might note that the times are all zero. That is because this example does a trivial amount of work, and each expression took less than a millisecond to complete. But it does illustrate some things that are useful when looking at the report:

- The `fn:current-dateTime()` function does not appear in the report, because profiling was disabled at this stage of the XQuery program.
- The expressions are not necessarily in the order they are in the XQuery program. If you want them in order, you can take the `prof:expression` elements and order them by the `prof:line` element (or one of the time elements, or whatever makes sense for your reports).
- While the expression 47 occurs while profiling is enabled, it does not show up in the output because it is just a constant, and constants do not appear in the profile report.

5.3.2 Returning a Part of the Profile Report

Because the profile report is XML, you can use XQuery to manipulate the report to suit your needs. The following example returns only a expression elements of the profile report, wraps it in an element, and orders it by the deep time element.

```
xquery version "1.0-ml";

let $req := xdmp:request()
let $dummy := prof:enable($req)
let $foo := for $i in fn:doc() return xdmp:node-uri($i)
let $dummy2 := prof:disable($req)
return <foo>{
   for $j in prof:report($req)//*:expression
     order by xs:dayTimeDuration($j/*:deep-time)
     return $j
}</foo>
```

To see the results, copy the code and run it against MarkLogic Server. You will need to enable profiling on the HTTP Server Configuration page, otherwise the report will be empty.

6.0 Disk Storage Considerations

This chapter describes how disk storage can affect performance of MarkLogic Server, and some of the storage options available for forests. It includes the following sections:

- Disk Storage and MarkLogic Server
- Fast Data Directory on Forests
- Large Data Directory on Forests

6.1 Disk Storage and MarkLogic Server

MarkLogic Server applications can be very disk-intensive in their system demands. It is therefore very important to size your hardware appropriate for your workload and performance requirements. The topic of disk storage performance is complicated; there are many factors that can influence performance including disk controllers, network latency, the speed and quality of the disks, and other disk technologies such as storage area networks (SANs) and solid state drive (SSD). As with most performance issues, there are price/performance trade-offs to consider.

For example, SSDs are quite expensive compared with rotating drives.

6.2 Fast Data Directory on Forests

In the forest configuration for each forest, you can configure a Fast Data Directory. The Fast Data Directory is designed for fast filesystems such as SSDs with built-in disk controllers. The Fast Data Directory stores the forest journals and as many stands as will fit onto the filesytem; if the forest never grows beyond the size of the Fast Data Directory, then the entire forest will be stored in that directory. If there are multiple forests on the same host that point to the same Fast Data Directory, MarkLogic Server divides the space equally between the different forests.

When the Fast Data Directory begins to approach its capacity, during periodic merges, MarkLogic Server will start to put data in the regular Data Directory. By specifying a Fast Data Directory, you can get much of the advantage of using the fast disk hardware while only buying a relatively small SSD (or other fast disk system). For example, consider a scenario where you have an 8-core MarkLogic Server d-host that is hosting 4 forests. If you have good quality commodity server-class rotating disk system with many magnetic disk spindles (for example, 6 disks in some RAID configuration) having 2 terabytes of storage, and if you have a 250 gigabyte SSD (for example, a PCI I/O accelerator card) for the fast data directory, then you can get a significant amount of the benefit of having the SSD storage while keeping the cost down (because the rotating storage is several times less expensive than the SSD storage). In this scenario, each of the 4 forests could us up to 1/4 of the size of the SSD, or about 62.5 GB. Once the forest size grows close to that limit, then the Data Directory with the rotating storage is used.

6.3 Large Data Directory on Forests

Just like you might want a different class of disk for the Fast Data Directory, you might also want a different class of disk for the Large Data Directory. The Large Data Directory stores binary document that are larger than the Large Size Threshold specified in the database configuration. This filesystem is typically a very large filesystem, and it may use a different class of disk than your regular filesystem (or it may just be on a different set of the same disks). For more details about binary documents, see Working With Binary Documents in the *Application Developer's Guide*.

7.0 Monitoring MarkLogic Server Performance

This chapter provides an overview to various ways to monitor the status in MarkLogic Server, both through the Admin Interface and through Server Monitoring APIs designed to report status of various parts of the system. This chapter includes the following sections:

- Ways to Monitor Performance and Activity
- Server Monitoring APIs

7.1 Ways to Monitor Performance and Activity

This section describes the following ways to monitor various activity on MarkLogic Server:

- Server Logs
- Status Screens in the Admin Interface
- Create Your Own Server Reports

7.1.1 Server Logs

The logs for MarkLogic Server are an important source of information about activity on the server, particularly information about error conditions. The logs are all stored in the `Logs` directory under the MarkLogic Server data directory (typically `c:\Program Files\MarkLogic\Data\Logs` in windows, `/var/opt/MarkLogic/Logs` under UNIX-based systems). There are two types of logs:

- `ErrorLog.txt`, which logs MarkLogic Server exceptions, startup activity, and so on.
- `<port_no>_AccessLog.txt`, which logs access requests (for example, HTTP requests) for the App Server running at the specified port.

You can configure how long to keep a log before starting a new one, at which level to log activity, and how many old log files to keep before deleting (they have a number appended to their name, for example, `ErrorLog_5.txt` indicates 5 new log files have been created since this one was used). For more details on configuring the log files, see the *Administrator's Guide*.

Another option you can configure, at the App Server level, is for any uncaught application exceptions to be written to the `ErrorLog.txt` file. This way, if your application throws an exception (for example, if it has a syntax error), the error message is logged in addition to being sent to the client. This is useful in debugging, especially if queries are being generated via user activity on a browser or through a WebDAV client.

You can also code your own log messages into an application using `xdmp:log`. You can use `xdmp:log` to log any message at any level, and that message is written to the `ErrorLog.txt` file when it is called. Log messages are useful in debugging during development, and are also useful in logging certain activities in production.

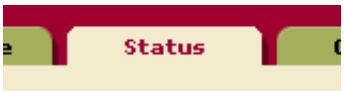
For operational purposes, some developers write scripts or programs to monitor the logs for specific messages. Then, if the specific message is logged, the script or program can send some sort of alert out (for example, page someone or send a message to someone).

The logs contain important information that can be used in monitoring MarkLogic Server. The logs can be a powerful tool in an overall monitoring policy. How you use that information depends on your requirements.

Note: There must be sufficient disk space on the filesystem in which the log files reside. If there is no space left on the log file device, MarkLogic Server will abort. Additionally, if there is no disk space available for the log files, MarkLogic Server will fail to start.

7.1.2 Status Screens in the Admin Interface

The Admin Interface includes many pages that list current activity and status for various parts of MarkLogic Server. To access the status pages, click the status tab in the part of the Admin Interface to which you want to find current system information.



The following table lists the status pages available in the Admin Interface, along with the location path to the status tab in the Admin Interface and a description of each page.

Name	Location	Description
System Status	Configure > Status	Provides a summary of all information throughout the entire MarkLogic Server cluster, including host, App Server, database, and forest information. Also includes buttons to restart or shutdown all of the hosts in the cluster.
Group Status	Groups > *group_name* > Status	Provides a summary of all information throughout the MarkLogic Server group, including host, App Server, database, and forest information. Also includes buttons to restart or shutdown all of the hosts in the group.

Name	Location	Description
Host Status	Hosts > *host_name* > Status	Provides a summary of the current conditions on the host, including information about App Servers, forests, and active queries. Also includes buttons to enter a new license key, restart, or shutdown the host.
App Server Status	Groups > *group_name* > App Servers > *app_server_name* > Status	Shows information about activity on the App Server, including queries active on each host and the ability to cancel the queries.
Task Server Status	Groups > *group_name* > Task Server > Status	Shows information about activity on the task servers for the group, including the number of requests being processed and the number of tasks queued on the task server.
Database Status	Databases > *db_name* > Status	Shows activity on the specified database. Shows if any merges are in progress and if reindexing is in progress, and gives estimates about how long they will take. Shows information about the database, including the number of documents, number of fragments, and its size. Optionally shows forest status information.
Forest Status	Forests > *forest_name* > Status	Shows information and activity about the specified forest, including merge activity, number of stands, size, space available, and so on. Also includes a button to restart the forest, which forces the forest to go offline. Once it is offline, it will automatically attempts to rejoin the host, resulting in a restart of the forest. If failover is enabled, a restarted forest will first attempt to join the primary host.

7.1.3 Create Your Own Server Reports

The status pages in the Admin Interface provide a lot of detail about many parts of the system. If you want information about MarkLogic Server status in a different form, however, or if you want to combine it with some other application-specific information, you can build your own server reports using the Server Monitoring APIs. For more details, see the next section.

7.2 Server Monitoring APIs

The Server Monitoring APIs are XQuery functions that return XML representations of current activity of various parts of MarkLogic Server. Because they are XQuery functions, you can build any application you deem necessary with them. For example, perhaps you want a summary page that shows some different information then the system status page in the Admin Interface, or perhaps you want to combine some of that information with some content from your application. Perhaps you want to integrate it with a site-wide monitoring system. Whatever your requirements, because the APIs return XML, it is easy to write an application to display the information in whatever way fits your needs.

The following are the monitoring functions available:

- `xdmp:forest-counts`
- `xdmp:forest-status`
- `xdmp:host-status`
- `xdmp:request-cancel`
- `xdmp:request-status`
- `xdmp:server-status`

The `xdmp:forest-counts` function can take some time to compute on large systems, as it must query the forest to determine some of the counts. You can limit the work it performs with the optional second argument. For more details about syntax and usage of these functions, see the Server Monitoring functions in the *MarkLogic XQuery and XSLT Function Reference*.

8.0 Technical Support

MarkLogic provides technical support according to the terms detailed in your Software License Agreement or End User License Agreement. For evaluation licenses, MarkLogic may provide support on an “as possible” basis.

For customers with a support contract, we invite you to visit our support website at http://support.marklogic.com to access information on known and fixed issues.

For complete product documentation, the latest product release downloads, and other useful information for developers, visit our developer site at http://developer.marklogic.com.

If you have questions or comments, you may contact MarkLogic Technical Support at the following email address:

support@marklogic.com

If reporting a query evaluation problem, please be sure to include the sample XQuery code.

Combined Product Notices

MarkLogic Server 6.0 and supporting products.
Last updated: August 19, 2013

COPYRIGHT

Copyright © 2013 MarkLogic Corporation. All rights reserved.
This technology is protected by U.S. Patent No. 7,127,469B2, U.S. Patent No. 7,171,404B2, U.S. Patent No. 7,756,858 B2, and U.S. Patent No 7,962,474 B2.

The MarkLogic software is protected by United States and international copyright laws, and incorporates certain third party libraries and components which are subject to the attributions, terms, conditions and disclaimers set forth below.

1. Contains SAP BusinessObjects Text Analysis XI from SAP AG. Copyright © 1996-2011. All rights reserved.

2. Highslide Software from Highslide Software Torstein Honsi. All Highslide Software is protected by local and international copyright laws. All rights reserved.

3. Icons developed by Yusuke Kamiyamane. Copyright © 2011 Yusuke Kamiyamane. All rights reserved. Icons are licensed subject to http://creativecommons.org/licenses/by/3.0/legalcode.

4. Antenna House OfficeHTML Copyright © 2000-2008 Antenna House, Inc. All rights reserved.

5. Argus Copyright ©1999-2008 Iceni Technology Ltd. All rights reserved.

6. Rosette Linguistics Platform 6.5.2/6.5.3 from Basis Technology Corporation, Copyright © 2004-2008 Basis Technology Corporation. All rights reserved.

7. ISYS Search. Copyright © 2011 ISYS™ Search Software, Inc. All rights reserved.

8. Software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/) Copyright © 1995-1998 Eric Young (eay@cryptsoft.com). All rights reserved. Copyright © 1998-2011 The OpenSSL Project. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. All advertising materials mentioning features or use of this software must display the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)" 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org. 5. Products derived from this

software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project. 6. Redistributions of any form whatsoever must retain the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/)" THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com). Additional terms may apply to the foregoing software as further set forth at http://www.openssl.org/source/license.html

9. ICU v. 4.2.1 available from http://site.icu-project.org// Copyright © 1995-2011 International Business Machines Corporation and others. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

10. Tidy available at http://www.w3.org/People/Raggett/tidy/. Copyright © 1998-2008 World Wide Web Consortium (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved.

11. 'zlib' general purpose compression library v. 1.2.3. Copyright © 1995-2010 Jean-loup Gailly and Mark Adler. http://www.zlib.net/zlib_license.html

12. RSA Data Security, Inc. MD5 Message-Digest Algorithm, copyright © 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved. License to copy and use this software is granted provided that it is identified as the "RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing this software or this function. License is also granted to make and use derivative works provided that such works are identified as "derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing the derived work. RSA Data Security, Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided "as is" without express or implied warranty of any kind. These notices must be retained in any copies of any part of this documentation and/or software.

13. TRE software, available at http://laurikari.net/tre/. Copyright © 2001-2009 Ville Laurikari <vl@iki.fi>. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: (1) Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. (2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

14. Eigen Library, version 2.0.10, available at http://eigen.tuxfamily.org/index.php?title=Main_Page. Eigen Library is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. Eigen Library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with Eigen Library. If not, see: http://www.gnu.org/copyleft/lesser.html

15. FunctX XQuery Library v.1.0 available at http://www.xqueryfunctions.com/xq/, Copyright © Datypic (Priscilla Walmsley). You can redistribute such library and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the

hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License as available at http://www.gnu.org/licenses/lgpl-2.1.html#SEC4 for more details.

16. DeployJava.js v. 1.8, Copyright © 2006, 2011, Oracle and/or its affiliates. All rights reserved. ORACLE PROPRIETARY/CONFIDENTIAL. Use of the foregoing library is subject to the following license terms. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of Oracle nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

17. YUI libraries v.2.8 available at http://developer.yahoo.com/yui/. Copyright © 2011 Yahoo! Inc. All rights reserved. Use of the foregoing libraries is subject to the following license terms. Redistribution and use of this software in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of Yahoo! Inc. nor the names of YUI's contributors may be used to endorse or promote products derived from this software without specific prior written permission of Yahoo! Inc. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING

NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Additional notices related to components included in the foregoing library are also available at the following URL: http://yuilibrary.com/license/

18. XSLTForms libraries available at http://sourceforge.net/projects/xsltforms/ . You can redistribute such libraries and/or modify them under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. These libraries are distributed in the hope that they will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License as available at http://www.gnu.org/licenses/lgpl-2.1.html#SEC4 for more details.

19. jQuery v.1.7 libraries available at http://jquery.com/ and jQuery UI v.1.8.2 libraries available at http://jqueryui.com/. Copyright © 2011 John Resig. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OFMERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE ANDNONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BELIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTIONOF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTIONWITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

20. JSON v. 2 libraries available at http://www.json.org/json2.js. Copyright © 2002 JSON.org. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. The Software shall be used for Good, not Evil. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

21. CodeMirror libraries available at http://codemirror.net/. Copyright © 2011 by Marijn Haverbeke marijnh@gmail.com. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

22. History.js available https://github.com/balupton/History.js/ Copyright © 2011, Benjamin Arthur Lupton. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of Benjamin Arthur Lupton nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

23. Sample content provided for use with the software is copyrighted by the respective authors as contributed by each of them to Wikipedia pursuant to the Creative Commons Attribution-ShareAlike License found at http://creativecommons.org/licenses/by-sa/3.0/.

24. Visual Studio Tools for Office 3.0 (Microsoft VSTO 3.0) licensed under Microsoft Public License Microsoft Public License (Ms-PL) found at http://msdn.microsoft.com/en-us/library/ff647676.aspx.

25. The MarkLogic Connector for Hadoop contains Apache Jakarta Commons Modeler. Copyright 2001-2007 The Apache Software Foundation. Licensed under the Apache License, Version 2.0 (the "License"). You may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

26. jemalloc http://www.canonware.com/jemalloc/index.html, released under the terms of the following BSD-derived licenses: Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice(s), this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice(s), this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER(S) ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER(S) BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Portions of the software are copyright (C) 2002-2013 Jason Evans, and portions copyright (C) 2009-2013 Facebook, Inc. and copyright (C) 2007-2012 Mozilla Foundation. Neither the name of Facebook, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. All rights reserved.

27. Gziphelper; version: 1.0, Feb, 2003, authored by Gao Dasheng, is distributed subject to the following terms: This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software. Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions: (1.) the origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required; (2) altered source versions must be plainly marked as such, and must not be misrepresented as being the original software; (3) this notice may not be removed or altered from any source distribution.

28. Some features of MarkLogic allow you to develop and use Google Maps. Review Google Map's privacy policy at http://www.google.com/policies/privacy/, legal notices at http://maps.google.com/help/legalnotices_maps.html, and terms of use at http://maps.google.com/help/terms_maps.html.

29. commons-codec, HttpComponents, commons-logging, and Google Maps Utility are used under the Licensed under the Apache License, Version 2.0 (the "License"). You may not use these files except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

30. Mapstraction. Copyright (c) 2012, Mapstraction. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

31. logback-classic and logback-core are provided in object code form only under the Eclipse Public License - v 1.0 available at http://www.eclipse.org/legal/epl-v10.html. Copyright (C) 1999-2012, QOS.ch. All rights reserved. Source code for the Program is available http://logback.qos.ch/.

32. Simple Logging Facade for Java (SLF4J). Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

33. cURL v. 7.22.0 available at http://curl.haxx.se/. Copyright © 1996 - 2011, Daniel Stenberg, <daniel@haxx.se>. All rights reserved. Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

34. heatmap.js distributed subject to the "THE BEER-WARE LICENSE" (Revision 42): <phk@FreeBSD.ORG> wrote this file. As long as you retain this notice you can do whatever you want with this stuff. If we meet some day, and you think this stuff is worth it, you can buy me a beer in return Poul-Henning Kamp.

35. Except for components that are otherwise subject to other open source licenses, the MarkLogic ODBC Driver is released under the GNU Lesser General Public License as published by the Free Software Foundation, version 2.1 of the License.

36. Solr-commons-csv-3.5.0 Copyright 2005-2006 The Apache Software Foundation. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

37. Avro v. 1.7.1. Copyright © The Apache Software Foundation. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

38. Guava v. 11.0.2. Copyright © Google. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

39. Hadoop-auth v. 2.0.0. Copyright © The Apache Software Foundation. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

40. hadoop-hdfs v. 2.0. Copyright © The Apache Software Foundation. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

41. commons-lang v. 2.4, 2.5. Copyright © The Apache Software Foundation. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

42. commons-configuration v. 2.0. Copyright © The Apache Software Foundation. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

43. jackson-core-asl v. 1.8.8. Copyright © FasterXML. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

44. jackson-mapper-asl v. 1.8.8. Copyright © FasterXML. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

45. log4j v. 1.2.16. Copyright © The Apache Software Foundation. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, this file is distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

46. protobuf-java v. 2.4.0a. Copyright © Google. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of Google nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

47. xpp3 v. 1.1.3.3. Copyright © 2002 Extreme! Lab, Indiana University. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Indiana University Extreme! Lab(http://www.extreme.indiana.edu/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. The names "Indiana Univeristy" and "Indiana Univeristy Extreme! Lab" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact http://www.extreme.indiana.edu/. Products derived from this software may not use "Indiana Univeristy" name nor may "Indiana Univeristy" appear in their name, without prior written permission of the Indiana University. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS, COPYRIGHT HOLDERS OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IFADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

48. xstream v. 1.4.2. Copyright © 2003-2006, Joe Walnes. Copyright © 2006-2009, 2011 XStream Committers. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of XStream nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR

BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

TRADEMARK NOTICE

The MarkLogic name and logo are registered trademarks of MarkLogic Corporation in the United States and/or other countries. Excel and PowerPoint are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Any other names or logos included in the Software, this notice or the documentation are property of the respective trademark owners.