MarkLogic Server

Loading Content Into MarkLogic Server

MarkLogic 6
September, 2012

Last Revised: 6.0-4, July, 2013

Copyright © 2013 MarkLogic Corporation. All rights reserved.

1.0 Designing a Content Loading Strategy 5
1.1 Available Content Loading Interfaces 6
1.2 Loading Activities 7
1.3 What to Consider Before Loading Content 7
1.3.1 Setting Document Permissions 8
1.3.2 Schemas 8
1.3.3 Fragments 8
1.3.4 Indexing 8

2.0 Controlling Document Format 9
2.1 Terminology 9
2.2 Supported Document Formats 10
2.2.1 XML Format 10
2.2.2 Binary Format 10
2.2.3 Text (CLOB) Format 11
2.3 Choosing a Binary Format 11
2.3.1 Loading Binary Documents 14
2.3.2 Configuring MarkLogic Server for Binary Documents 14
2.4 Implicitly Setting the Format Based on the MIME Type 14
2.5 Explicitly Setting the Format 14
2.6 Determining the Format of a Document 15

3.0 Specifying Encoding and Language 17
3.1 Understanding Character Encoding 17
3.2 Explicitly Specifying Character Encoding While Loading 18
3.3 Automatically Detecting the Encoding 19
3.4 Inferring the Language and Encoding of a Node in XQuery with xdmp:encoding-language-detect 19
3.5 Specifying the Default Language for XML Documents 20

4.0 Loading Content Using Information Studio 22

5.0 Loading Content Using XQuery 23
5.1 Built-In Document Loading Functions 23
5.2 Specifying a Forest in Which to Load a Document 24
5.2.1 Advantages of Specifying a Forest 24
5.2.2 Example: Examining a Document to Decide Which Forest to Specify 24
5.2.3 More Examples 25
5.3 Loading External Binaries Using XQuery 26

6.0 Loading Content Using the REST API and the Java API 27

7.0 Using MarkLogic Content Pump 28

7.1 MarkLogic Content Pump Overview 28
7.2 Terms and Definitions 29
7.3 Modifying the Example Commands for Windows 30
7.4 Downloading and Installing mlcp 30
7.4.1 Supported Platforms 30
7.4.2 Required Software 31
7.4.3 Installing mlcp 31
7.5 Security Considerations 31
7.6 Using mlcp with a Multi-Host MarkLogic Cluster 32
7.6.1 All Forest Hosts Must Be Reachable 32
7.6.2 Export and Copy Require In-Forest Evaluation of Queries 33
7.6.3 Cluster-Wide XDBC Configuration Requirements 33
7.7 Configuring Distributed Mode 34
7.7.1 Specifying the Hadoop Configuration File Location 34
7.7.2 Setting Custom Hadoop Options and Properties 35
7.7.3 Required Hadoop User Privileges 35
7.8 Importing Content Into MarkLogic Server 35
7.8.1 Supported Input Format Summary 36
7.8.2 Understanding Input File Path Resolution 37
7.8.3 Controlling Database URIs During Ingestion 37
7.8.3.1 Default Document URI Construction 38
7.8.3.2 Transforming the Default URI 40
7.8.3.3 Character Encoding of URIs 40
7.8.4 How mlcp Determines Document Type 41
7.8.5 Loading Documents from a Directory 42
7.8.5.1 Loading a Single File 43
7.8.5.2 Loading All the Files in a Directory 43
7.8.5.3 Filtering Documents Loaded From a Directory 44
7.8.6 Loading Documents From Compressed Files 44
7.8.7 Loading Content and Metadata From an Archive 45
7.8.8 Splitting Large XML Files Into Multiple Documents 46
7.8.9 Creating Documents from Delimited Text Files 49
7.8.10 Creating Documents from Hadoop Sequence Files 49
7.8.10.1 Basic Steps 50
7.8.10.2 Implementing the Key and Value Interfaces 50
7.8.10.3 Deploying your Key and Value Implementation 52
7.8.10.4 Loading Documents From Your Sequence Files 52
7.8.10.5 Running the SequenceFile Example 52
7.8.11 Performance Considerations for Loading Documents 53
7.8.11.1 Time vs. Space: Configuring Batch and Transaction Size 54
7.8.11.2 Time vs. Correctness: Using Direct Forest Updates 54
7.8.11.3 Tuning Split Size for Local Mode 55
7.8.11.4 Tuning Split Size for Distributed Mode 56
7.8.11.5 Reducing Memory Consumption With Streaming 56
7.9 Controlling What is Exported or Copied 57
7.9.1 Filtering Document Exports 57

7.9.2 Filtering Archive and Copy Contents 57
7.10 Exporting Content from MarkLogic Server 58
7.10.1 Potential Export Output Inconsistency 58
7.10.2 How URI Decoding Affects Output File Names 58
7.10.3 Exporting Documents as Files 59
7.10.4 Exporting Documents to a Compressed File 59
7.10.5 Exporting to an Archive 60
7.11 Copying Content Between Databases 61
7.12 Command Line Reference 63
7.12.1 Command Line Summary 63
7.12.2 Setting Java Virtual Machine (JVM) Options 64
7.12.3 Import Options 64
7.12.4 Export Options 70
7.12.5 Copy Options 73
7.12.6 Regular Expression Syntax 76
7.12.7 Options File Syntax 77

8.0 Loading Content Using WebDAV 79

9.0 Repairing XML Content During Loading 80
9.1 Programming Interfaces and Supported Content Repair Capabilities 81
9.2 Enabling Content Repair 81
9.3 General-Purpose Tag Repair 82
9.3.1 How General-Purpose Tag Repair Works 82
9.3.2 Pitfalls of General-Purpose Tag Repair 83
9.3.3 Limitations 84
9.3.3.1 XQuery Functions 84
9.3.3.2 Root Element 84
9.3.3.3 Previous Marklogic Versions 84
9.3.4 Controlling General-Purpose Tag Repair 85
9.4 Auto-Close Repair of Empty Tags 85
9.4.1 What Empty Tag Auto-Close Repair Does 85
9.4.2 Defining a Schema to Support Empty Tag Auto-Close Repair 86
9.4.3 Invoking Empty Tag Auto-Close Repair 87
9.4.4 Scope of Application 89
9.4.5 Disabling Empty Tag Auto-Close 90
9.5 Schema-Driven Tag Repair 90
9.5.1 What Schema-Driven Tag Repair Does 90
9.5.2 How to Invoke Schema-Driven Tag Repair 92
9.5.3 Scope of Application 93
9.5.4 Disabling Schema-Driven Tag Repair 93
9.6 Load-Time Default Namespace Assignment 94
9.6.1 How Default Namespace Assignments Work 94
9.6.2 Scope of Application 95
9.7 Load-Time Namespace Prefix Binding 95

9.7.1 How Load-Time Namespace Prefix Binding Works 96
9.7.2 Interaction with Load-Time Default Namespace Assignment 97
9.7.3 Scope of Application 99
9.7.4 Disabling Load-Time Namespace Prefix Binding 99
9.8 Query-Driven Content Repair 99
9.8.1 Point Repair 100
9.8.2 Document Walkers 100

10.0 Modifying Content During Loading 103
10.1 Converting Microsoft Office and Adobe PDF Into XML 103
10.2 Information Studio Transforms 103
10.3 Converting to XHTML 104
10.4 Automating Metadata Extraction 104
10.5 Transforming XML Structures 104

11.0 Performance Considerations 105
11.1 Understanding the Locking and Journaling Database Settings for Bulk Loads .105
11.2 Fragmentation 106

12.0 Technical Support 107

Combined Product Notices 108
COPYRIGHT 108
TRADEMARK NOTICE 120

1.0 Designing a Content Loading Strategy

MarkLogic Server provides many ways to load content into a database including built-in XQuery functions, the GUI-based interface of MarkLogic Information Studio, and the command-line tool, MarkLogic Content Pump (mlcp). Choosing the appropriate method for a specific use case depends on many factors, including the characteristics of your content, the source of the content, the frequency of loading, and whether the content needs to be repaired or modified during loading. In addition, environmental and operational factors such as workflow integration, development resources, performance considerations, and developer expertise often need to be considered in choosing the best tools and mechanisms.

The MarkLogic mechanisms for loading content provide varying trade-offs along a number of dimensions such as the following:

- Usability and flexibility of the interface itself
- Performance, scalability, I/O capacity
- Loading frequency
- Automation or scripting requirements
- Workflow and integration requirements

This chapter lists the various tools to load content and contains the following sections:

- Available Content Loading Interfaces
- Loading Activities
- What to Consider Before Loading Content

1.1 Available Content Loading Interfaces

There are several ways to load content into MarkLogic Server. The following table summarizes content loading interfaces and their benefits.

Interface/Tool	Description	Benefits
MarkLogic Connector for Hadoop	A set of Java classes that enables loading content from HDFS into MarkLogic Server.	Distributed processing of large amounts of data
MarkLogic Content Pump (mlcp)	A command line tool	Ease of workflow integration, can leverage Hadoop processing, bulk loading of billions of local files, split and load aggregate XML or delimited text files
Information Studio	A browser UI for creating and loading content into databases. Features include collecting input data from multiple sources, applying transformations, and metadata extraction.	Ease of use, does not require XQuery or other programming skills, allows for simple transformations
MarkLogic Java API	A set of Java classes for supporting document manipulation and search operations	Leverage existing Java programming skills
MarkLogic REST API	A set of REST services hosted on an HTTP application server and associated with a content database that enable developers to build applications on top of MarkLogic Server without writing XQuery	Leverage existing REST programming skills
XCC	XML Contentbase Connector (XCC) is an interface to communicate with MarkLogic Server from a Java or .NET middleware application layer	Create multi-tier applications with MarkLogic Server as the underlying content repository
XQuery API	An extensive set of XQuery functions that provides maximum control	Flexibility and expanded capabilities

1.2 Loading Activities

There are various things you can do with each of the loading interfaces, all resulting in ingesting data into the database. The following are some of the things you might do through the interfaces. Which interface you use is a matter of which you are most comfortable with as well as trade-offs that some might have over others (for example, ease-of-use versus extensibility). While each tool can usually accomodate each of these activities, in cases where one tool has a specific feature to make one of these activites easy, it is called out in the list.

- Load from a directory (mlcp)
- Load from compressed files (mlcp)
- Split single aggregate XML file into multiple documents (mlcp)
- Load large numbers of small files
- Load delimited text files (mlcp)
- One document in > one document out (Information Studio)
- Enrich the documents
- Extract information from the documents (metadata, new elements)
- Extract some information and load only extracted information
- Load large binary files
- Create neutral format archive (mlcp)
- Copy from one ML database to another ML database (mlcp)

1.3 What to Consider Before Loading Content

Designing your content loading strategy depends on the complexity of your source content, the nature of the output to be inserted into the database and many other factors. This section lists some of the areas to think about with links to more detailed discussions and contains the following parts:

- Setting Document Permissions
- Schemas
- Fragments
- Indexing

1.3.1 Setting Document Permissions

When you load documents into a database, be sure you either explicitly set permissions in the document loading API or have configured default permissions on the user (or on roles for that user) who is loading the documents. Default permissions are applied to a document when it is loaded if you do not explicitly set permissions.

Permissions on a document control access to capabilities (`read`, `insert`, `update`, and `execute`) on the document. Each permission consists of a capability and a corresponding role. To have a specific capability for a document, a user must have the role paired with that capability on the document permission. Default permissions are specified on roles and on users in the Admin Interface.

If you load a document without the needed permissions, users might not be able to read, update, or execute the document (even by the user who loaded the document). For an overview of security, see *Understanding and Using Security Guide*. For details on creating privileges and setting permissions, see the Security Administration chapter of the *Administrator's Guide*.

Note: When you load a document, be sure that a named role has update permissions. For any document created by a user who does not have the `admin` role, the document must be created with at least one update permission or MarkLogic throws an `XDMP-MUSTHAVEUPDATE` exception during document creation. If there is no role on a document's permissions with update capability, or if the document has no permissions, then only users with the `admin` role can update or delete the document.

1.3.2 Schemas

Schemas are automatically invoked by the server when loading documents (for conducting content repair) and when evaluating queries (for proper data typing). If you plan to use schemas in your content loading strategy, review the information in the Loading Schemas chapter in the *Application Developer's Guide*.

1.3.3 Fragments

When loading data into a database, you have the option of specifying how XML documents are partitioned for storage into smaller blocks of information called fragments. For large XML documents, size can be an issue, and using fragments may help manage performance of your system. For a discussion of fragments, see Fragments in the *Administrator's Guide*.

1.3.4 Indexing

Before loading documents into a database, you have the option of specifying a number of parameters that impact how the text components of those documents are indexed. These settings can affect query performance and disk usage. For details, see Text Indexing in the *Administrator's Guide*.

2.0 Controlling Document Format

Each document in a MarkLogic Server database has a *format* associated with it. The format is based on the root node of the document. Once a document has been loaded as a particular format, you cannot change the format unless you replace the root node of the document with one of a different format. You can replace the root node of a document to change its format in a number of ways, including reloading the document while specifying a different format, deleting the document and then loading it again with the same URI, or replacing the root node with one of a different format.

Documents loaded into a MarkLogic Server database in XML or text format are always stored in UTF-8 encoding. Documents loaded in XML or text format must either already be in UTF-8 encoding or the UTF-8 encoding must be explicitly specified during loading using options available in the load APIs. For example, you might use the encoding option of the `xdmp:document-load` function. For more details, see Encodings and Collations in the *Search Developer's Guide*.

The following topics are included:

- Supported Document Formats
- Choosing a Binary Format
- Implicitly Setting the Format Based on the MIME Type
- Explicitly Setting the Format
- Determining the Format of a Document

2.1 Terminology

The following terms are used in this topic.

Term	Definition
document format	Refers to how documents are stored in MarkLogic databases: XML, binary, or text format.
QName	QName stands for qualified name and defines a valid identifier for elements and attributes. QNames are used to reference particular elements or attributes within XML documents.
small binary document	A binary document stored in a MarkLogic database whose size does not exceed the large size threshold. For details, see "Choosing a Binary Format" on page 11.

Term	Definition
large binary document	A binary document stored in a MarkLogic database whose size exceeds the large size threshold. For details, see “Choosing a Binary Format” on page 11.
external binary document	A binary document that is not stored in a MarkLogic database and whose contents are not managed by the server. For details, see “Choosing a Binary Format” on page 11.
CLOB	Character large object documents, or text documents.
BLOB	Binary large object documents, binary data stored as a single entity. Typically images, audio, or other multimedia object.

2.2 Supported Document Formats

MarkLogic supports the following document formats:

- XML Format
- Binary Format
- Text (CLOB) Format

2.2.1 XML Format

Documents that are in XML format have special characteristics that enable you to do more with them. For example, you can use XPath expressions to search through to particular parts of the document and you can use the whole range of `cts:query` constructors to perform fine-grained search operations, including element-level search.

XML documents are indexed when they are loaded. The indexing speeds up query response time. The type of indexing is controlled by the configuration options set on your document’s destination database. One technique for loading extremely large XML documents is to fragment the documents using various elements in the XML structure. The maximum size of a single XML fragment is 128 MB for 32-bit machines, 512 MB for 64-bit machines. For more details about fragmenting documents, see Fragments in the *Administrator’s Guide*.

2.2.2 Binary Format

Binary documents are loaded into the database as binary nodes. Each binary document is a single node with no children. Binary documents are typically not textual in nature and require another application to read them. Some typical binary documents are image files (for example, `.gif`, `.jpg`), Microsoft Word files (`.doc` and `.docx`), executable program files, and so on.

Binary documents are not indexed when they are loaded.

MarkLogic Server supports three kinds of binary documents: small, large (BLOBs), and external. Applications use the same interfaces to read all three kinds of binary documents, but they are stored and loaded differently. These differences may lead to tradeoffs in access times, memory requirements, and disk consumption. For more details, see "Choosing a Binary Format" on page 11.

For a discussion of the sizing and configuration options to consider when working with binary content, see Configuring MarkLogic Server for Binary Content in the *Application Developer's Guide*.

2.2.3 Text (CLOB) Format

Character large object (CLOB) documents, or *text* documents, are loaded into the database as text nodes. Each text document is a single node with no children. Unlike binary documents, text documents are textual in nature, and you can therefore perform text searches on them. Because text documents only have a single node, however, you cannot navigate through the document structure using XPath expressions like you can with XML documents.

Some typical text documents are simple text files (`.txt`), source code files (`.cpp`, `.java`, and so on), non well-formed HTML files, or any non-XML text file.

For 64-bit machines, text documents have a 64 MB size limit. The `in memory tree size` limit database property (on the database configuration screen in the Admin Interface) should be at least 1 or 2 megabytes larger than the largest text document you plan on loading into the database.

The database text-indexing settings apply to text documents (as well as XML documents), and MarkLogic creates the indexes when the text document is loaded.

2.3 Choosing a Binary Format

Binary documents require special consideration because they are often much larger than text or XML content. MarkLogic Server supports three types of binary documents: *small*, *large*, and *external*. Applications use the same interfaces to read all three types of binary document, but they are stored and loaded differently. A database may contain any combination of small, large, and external binaries. Choose the format that best matches the needs of your application and the capacity of your system. The size threshold that defines small and large binary objects is configurable. For details, see Selecting a Location For Binary Content in the *Application Developer's Guide*.

The following table summarizes attributes you should consider when organizing binary content:

Binary Type	Managed By MarkLogic Server	Stored In	Considerations
Small	Yes	Stands	• Fully cached for faster access • Entire contents may be cached in memory when accessed • Size and quantity constrained by available memory • Best suited for small frequently accessed content, such as thumbnails, profile photos, and icons
Large	Yes	Large Data Directory	• Access times similar to file system reads • Cached in compressed chunks for efficient resource utilization. • Streams documents into and out of the database • Size and quantity limited only by disk space and system file size limit • Best suited for movies, music, and high definition images
External	No	File system	• Access times similar to file system reads • Cached in compressed chunks for efficient resource utilization. • Streams documents into and out of the database • Size and quantity limited only by disk space and system file size limit • External contents do not participate in transactions, backups, or replication • Best suited for read-only content managed external to MarkLogic Server

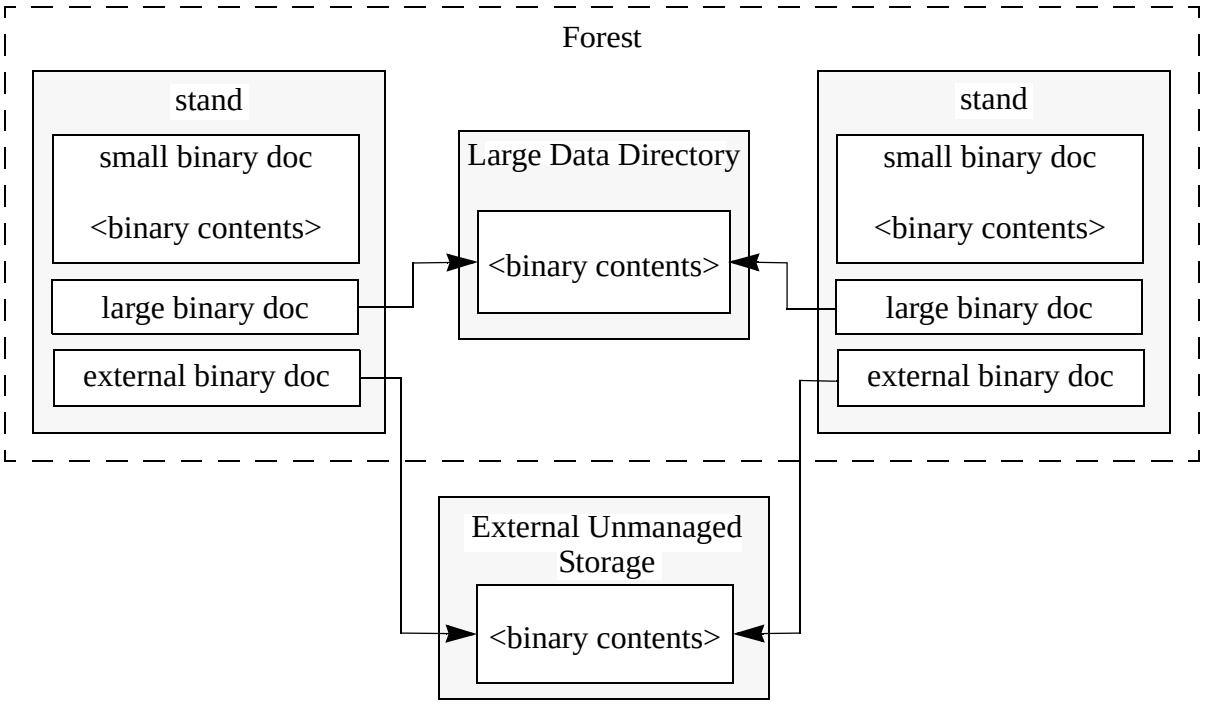
Small and large binary documents are stored in a MarkLogic database and are fully managed by MarkLogic Server. These documents fully participate in transactions, backup, and replication. Small binaries are stored directly in the stands of a forest, which means they are cached in memory. Large binaries are stored in a special Large Data Directory, with only a small reference object in the stand. The data directory containing large binary documents is located inside the forest by default. The location is configurable during forest creation. For more details, see Selecting a Location For Binary Content.

MarkLogic stores small and large binaries differently in the database to optimize resource utilization. For example, if multiple stands contain the same large binary document, only the reference fragment is duplicated. Similarly, if a new large binary document is created from a segment of an existing binary document using `xdmp:subbinary`, a new reference fragment is created, but the binary content is not duplicated. For details about stands, see Understanding Forests in the *Administrator's Guide*.

MarkLogic Server does not fully manage external binary documents because the documents are not stored in the database. The MarkLogic database contains only a small reference fragment to each external file. MarkLogic Server manages the reference fragments as usual, but does not manage the external files. For example, MarkLogic Server does not replicate or back up the external files. You must provide security, integrity, and persistence of the external files using other means, such as the underlying operating system or file system.

Large and external binary documents require little additional disk space for merges. During a merge, MarkLogic copies fragments from the old stands to a new merged stand, as described in Understanding and Controlling Database Merges in the *Administrator's Guide*. The small reference fragments of large and external binaries contribute little overhead to the merge process. The referenced binary contents are not copied during a merge.

The following diagram shows the differences in small, large, and external binaries handling. Although multiple stands may contain references fragments for the same large or external binary document, only the reference fragment is duplicated:



2.3.1 Loading Binary Documents

Loading small and large binary documents into a MarkLogic database does not require special handling, other than potentially explicitly setting the document format. Use the standard methods, such as Information Studio, XQuery functions, or other interfaces.

External binaries require special handling at load time because they are not managed by MarkLogic. For details, see "Loading External Binaries Using XQuery" on page 26.

2.3.2 Configuring MarkLogic Server for Binary Documents

Before loading binary content, you should carefully consider the sizing and scalability implications of binary documents and configure the server appropriately. For details, see Configuring MarkLogic Server for Binary Content in the *Application Developer's Guide*.

2.4 Implicitly Setting the Format Based on the MIME Type

Unless the format is explicitly set when you load a document, the format of the document is determined based on the MIME type that corresponds to the URI extension of the new document. The URI extension MIME types, along with their default formats, are set in the Mimetypes section of the Admin Interface. For example, with the default MIME type settings, documents loaded with the `xml` URI extension are loaded as XML files; therefore loading a document with a URI `http://marklogic.com/file.xml` results in loading an XML document.

You can also use the Mimetypes configuration page of the Admin Interface to modify any of the default content setting, create new MIME types, or add new extensions and associate a format. For example, if you know that all of your HTML files are well-formed (or clean up nicely with content repair), you might want to change the default content loading type of URIs ending with `.html` and `.htm` to XML.

2.5 Explicitly Setting the Format

When you load a document, you can explicitly set the format. Explicitly setting the format overrides the default settings specified on the Mimetypes configuration screen in the Admin Interface. For example, HTML files have a default format of text, but you might have some HTML files that you know are well-formed, and can therefore be loaded as XML.

> **Note:** It is a good practice to explicitly set the format rather than relying on implicit format settings based on the MIME types because it gives you complete control over the format and eliminates surprises based on implicit MIME type mappings.

The following table summarizes the mechanisms available for explicitly setting the document format during loading. For details, see the MarkLogic interface-specific documentation.

Interface	Summary	For More Details
Content Pump (mlcp)	Set the `-document_type` import option	"Importing Content Into MarkLogic Server" on page 35.
Information Studio	Set the ingestion format configuration option.	Configuring Ingestion Options in the *Information Studio Developer's Guide*.
Java API	`ContentDescriptor` interface.	*Java Application Developer's Guide*.
MarkLogic Connector for Hadoop	`ContentOutputFormat` class	*MarkLogic Connector for Hadoop Developer's Guide* and javadoc.
REST API	Set the `format` parameter or `Content-type` header on a PUT or POST request to the `/documents` service.	Loading Content into the Database in *REST Application Developer's Guide*, and the API reference for `PUT /v1/documents` or `POST /v1/documents` in the *REST Resources API*.
XCC	Set the format in the `ContentCreateOptions` class.	XCC Javadoc.
XQuery	Specify a value for the `<format>` element of the `<options>` node passed to `xdmp:document-load`.	The API documentation for `xdmp:document-load` in the *MarkLogic XQuery and XSLT Function Reference*.

The following XQuery example demonstrates explicitly setting the format to XML when using `xdmp:document-load`:

```
xdmp:document-load("c:\myFiles\file.html",
          <options xmlns="xdmp:document-load">
            <uri>http://myCompany.com/file.html</uri>
            <permissions>{xdmp:default-permissions()}</permissions>
            <collections>{xdmp:default-collections()}</collections>
            <format>xml</format>
          </options>)
```

2.6 Determining the Format of a Document

After a document is loaded into a database, you cannot assume the URI accurately reflects the content format. For example, a document can be loaded as XML even if it has a URI that ends in `.txt`. To determine the format of a document in a database, perform a node test on the root node of the document.

XQuery includes node tests to determine if a node is text (`text()`) or if a node is an XML element (`element()`). MarkLogic Server has added a node test extension to XQuery to determine if a node is binary (`binary()`).

The following code sample shows how you can use a `typeswitch` to determine the format of a document.

```
(: Substitute in the URI of the document you want to test :)
let $x:= doc("/my/uri.xml")/node()
return
typeswitch ( $x )
 case element() return "xml element node"
 case text() return "text node"
 case binary() return "binary node"
default return "don't know"
```

3.0 Specifying Encoding and Language

You can specify the encoding and default language while loading a document. You can also automatically detect the encoding or manually detect the language (for example, using `xdmp:encoding-language-detect`). This section describes how to load documents with a specific encoding or language, and includes the following parts:

- Understanding Character Encoding
- Explicitly Specifying Character Encoding While Loading
- Automatically Detecting the Encoding
- Inferring the Language and Encoding of a Node in XQuery with xdmp:encoding-language-detect
- Specifying the Default Language for XML Documents

For more information about languages, see Language Support in MarkLogic Server in the *Search Developer's Guide*.

3.1 Understanding Character Encoding

MarkLogic Server stores non-binary content in the UTF-8 encoding. MarkLogic Server automatically transcodes the content from the input encoding to UTF-8 during loading. All MarkLogic Server interfaces for ingestion support an encoding option or configuration parameter.

When loading non-binary content with an encoding other than UTF-8, use the option or configuration parameter to specify the input encoding. If the content cannot be converted to UTF-8, MarkLogic Server throws an exception. Specifying an encoding that does not match the input documents can lead to unexpected results.

Note the following about character encodings and conversion:

- MarkLogic Server always converts non-binary content into UTF-8.
- If no explicit encoding is specified through options or HTTP headers, the encoding defaults to UTF-8.
- If no explicit encoding option is specified, but there is an HTTP header specifying the encoding, then that encoding is used.
- If the encoding is UTF-8, an exception is thrown if the content contains non-UTF-8 characters.
- MarkLogic Server assumes the character set you specify is actually the character set of the content. If you specify an encoding that is different from the actual encoding of the characters, the result is undefined: An exception might be thrown during loading, or the invalid characters may be translated into the specified encoding incorrectly.

3.2 Explicitly Specifying Character Encoding While Loading

The table below summarizes the mechanisms available for explicitly specifying character encoding. See the interface-specific documentation for details. If no encoding is specified, MarkLogic Server defaults to UTF-8 for all non-binary documents.

Interface	Method	For Details See
MarkLogic Connector for Hadoop	The configuration property `mapreduce.marklogic.output.content.encoding`.	Configuring a MapReduce Job in the *MarkLogic Connector for Hadoop Developer's Guide* `MarkLogicConstants` in the *MarkLogic Hadoop MapReduce Connector API*
MarkLogic Content Pump (mlcp)	Character encoding cannot be controlled. Only UTF-8 is supported.	"Using MarkLogic Content Pump" on page 28.
MarkLogic Java API	Various handles.	Conversion of Document Encoding in the *Java Application Developer's Guide*
MarkLogic REST API	The `charset` parameter of the HTTP `Content-type` header. However, Text, XML and JSON content must be UTF-8 encoded.	*REST Application Developer's Guide*
XCC	Java: `ContentCreateOptions.setEncoding` XCC: `Encoding` property of the `ContentCreateOptions` class	Javadoc for XCC dotnet for XCC (C# API)
XQuery	The `encoding` element of the `options` parameter to `xdmp:document-load`, `xdmp:document-get`, `xdmp:zip-get`, and `xdmp:http-get`.	*XQuery and XSLT Reference Guide*

The following XQuery example loads the document using the ISO-8859-1 encoding, transcoding the content from ISO-8859-1 to UTF-8 during the load:

```
xdmp:document-load("c:/tmp/my-document.xml",
  <options xmlns="xdmp:document-load">
    <uri>/my-document.xml</uri>
    <encoding>ISO-8859-1</encoding>
  </options>)
```

3.3 Automatically Detecting the Encoding

For those interfaces that support auto-detection of encoding, MarkLogic Server attempts to automatically detect the encoding of non-binary content during loading if the explicitly specified encoding is `auto`.

The automatic encoding detection chooses an encoding equivalent to the first encoding returned by the `xdmp:encoding-language-detect` XQuery function. Encoding detection is not an exact science. There are cases where content encoding is ambiguous, but as long as your document is not too small, the encoding detection is fairly accurate. There are, however, cases where auto-detect might choose the wrong encoding.

The following XQuery example demonstrates using automatic character encoding detection when loading a document using `xdmp:document-load`:

```
xdmp:document-load("c:/tmp/my-document.xml",
  <options xmlns="xdmp:document-load">
    <uri>/my-document.xml</uri>
    <encoding>auto</encoding>
  </options>)
```

For details, see the interface specific documentation or “Explicitly Specifying Character Encoding While Loading” on page 18.

3.4 Inferring the Language and Encoding of a Node in XQuery with xdmp:encoding-language-detect

If you do not want to rely on the automatic detection for the encoding or if you want to detect the language, you can use the `xdmp:encoding-language-detect` function. The `xdmp:encoding-language-detect` function returns XML elements, each of which specifies a possible encoding and language for the specified node. Each element also has a score, and the one with the highest score (the first element returned) has the most likely encoding and language.

```
xdmp:encoding-language-detect(
  xdmp:document-get("c:/tmp/session-login.css"))
=>
<encoding-language xmlns="xdmp:encoding-language-detect">
  <encoding>utf-8</encoding>
  <language>en</language>
  <score>14.91</score>
</encoding-language>
<encoding-language xmlns="xdmp:encoding-language-detect">
  <encoding>utf-8</encoding>
  <language>ro</language>
  <score>13.47</score>
</encoding-language>
<encoding-language xmlns="xdmp:encoding-language-detect">
  <encoding>utf-8</encoding>
  <language>it</language>
  <score>12.84</score>
```

```
</encoding-language>
<encoding-language xmlns="xdmp:encoding-language-detect">
  <encoding>utf-8</encoding>
  <language>fr</language>
  <score>12.71</score>
</encoding-language>
...
```

The encoding detection is typically fairly accurate when the score is greater than 10. The language detection tends to be less accurate, however, because it can be difficult to detect the difference between some languages. Because it gives you the raw data, you can use the output from `xdmp:encoding-language-detect` with whatever logic you want to determine the language. For example, if you happen to know, based on your knowlege of the content, that the language is either Italian or Spanish, you can ignore entries for other languages.

Sometimes the language or the encoding of a block of text is ambiguous, therefore detecting languages and encodings is sometimes prone to error. As a rule, the larger the block of text, the higher the accuracy of the detection. If the size of the block of text you pass into xdmp:encoding-language-detect is more than a few paragraphs of text (several hundred bytes), then the detection is typically fairly accurate.

3.5 Specifying the Default Language for XML Documents

The formal or natural language of XML content is determined by the element attribute `xml:lang`. The language affects how MarkLogic Server tokenizes content, and therefore affects searching and indexing.

When there is no explicit `xml:lang` attribute on an XML document when it is loaded, MarkLogic Server uses the configured default language for the database. Set the database-wide default language through the `language` setting in the Admin UI.

You can override the configured default language for the database using load options, as shown by the table below:

Interface	Method	For Details See
MarkLogic Connector for Hadoop	The configuration property `mapreduce.marklogic.output.content.language`.	Configuring a MapReduce Job in the *MarkLogic Connector for Hadoop Developer's Guide* `MarkLogicConstants` in the *MarkLogic Hadoop MapReduce Connector API*
MarkLogic Content Pump	`-output_language` command line option	"Import Options" on page 64
MarkLogic Java API	`JSONDocumentManager` interface	*Java Application Developer's Guide* and javadoc
MarkLogic REST API	Set the `lang` parameter on a PUT or POST request to the `/documents` service.	`/documents` service and *REST Application Developer's Guide*
XCC	Java: `ContentCreateOptions.setLanguage` .NET: `Encoding` property of the `ContentCreateOptions` class.	Javadoc for XCC dotnet for XCC (C# API)
XQuery	Set the `<default-language>` element of the `<options>` node passed to `xdmp:document-load`, `xdmp:document-get`, `xdmp:http-get`, or `xdmp:zip-get`.	*XQuery and XSLT Reference Guide*

For details on languages, see Language Support in MarkLogic Server in the *Search Developer's Guide*.

4.0 Loading Content Using Information Studio

Information Studio is a browser-based interface that enables you to quickly create MarkLogic Server databases and load them with content. Information Studio comes with plugins, called collectors, that gather the content to load into a database. Specific collector implementations gather content in different ways. The following collectors are shipped with MarkLogic Server:

- The Filesystem Directory collector loads files from a specified directory. See Using the Filesystem Directory Collector in the *Information Studio Developer's Guide*.
- The Browser Drop-Box collector loads files dropped into a browser window. See Using the Browser Drop-Box Collector in the *Information Studio Developer's Guide*.
- The External Binary Filesystem Directory collector loads files from a specified directory as external binary documents. See Using the External Binary Filesystem Collector in the *Information Studio Developer's Guide*.

You can also create custom collectors as described in Creating Custom Collectors in the *Information Studio Developer's Guide*.

5.0 Loading Content Using XQuery

This chapter describes the XQuery interface for loading content and includes the following sections:

- Built-In Document Loading Functions
- Specifying a Forest in Which to Load a Document
- Loading External Binaries Using XQuery

5.1 Built-In Document Loading Functions

The `xdmp:document-load`, `xdmp:document-insert`, and `xdmp:document-get` functions can all be used as part of loading documents into a database. The `xdmp:document-load` function allows you to load documents from the filesystem into the database. The `xdmp:document-insert` function allows you to insert an existing node into a document (either a new or an existing document). The `xdmp:document-get` function loads a document from disk into memory. If you are loading a new document, the combination of `xdmp:document-get` and `xdmp:document-insert` is equivalent to `xdmp:document-load` of a new document.

Note: You may only load external binary documents using `xdmp:document-insert` of a constructed `external-binary` node. For details, see "Loading External Binaries Using XQuery" on page 26.

Note: The version 2.x `xdmp:load` and `xdmp:get` functions are deprecated in the current version of MarkLogic Server; in their place, use the `xdmp:document-load` and `xdmp:document-get` functions.

The basic syntax of `xdmp:document-load` is as follows:

```
xdmp:document-load(
   $location as xs:string,
   [$options as node()]
) as empty-sequence()
```

The basic syntax of `xdmp:document-insert` is as follows:

```
xdmp:document-insert(
   $uri as xs:string],
   $root as node()
   [$permissions as element(sec:permission)*],
   [$collections as xs:string*],
   [$quality as xs:integer],
   [$forest-ids as xs:unsignedLong*]
) as empty-sequence()
```

The basic syntax of `xdmp:document-get` is as follows:

```
xdmp:document-get(
   $location as xs:string],
   [$options as node()]
) as xs:node()
```

See the *XQuery and XSLT Reference Guide* for a more detailed syntax description.

5.2 Specifying a Forest in Which to Load a Document

When loading a document, you can use the `<forests>` node in an options node for `xdmp:document-load`, or the `$forest-id` argument to `xdmp:document-insert` (the sixth argument) to specify one or more forests to which the document is loaded. Specifying multiple forest IDs loads the document into one of the forests specified; the system decides which one of the specified forests to load the document. Once the document is loaded into a forest, it stays in that forest unless you delete the document, reload it specifying a different forest, or clear the forest.

Note: In order to load a document into a forest, the forest must exist and be attached to the database into which you are loading. Attempting to load a document into a forest that does not belong to the context database will throw an exception.

5.2.1 Advantages of Specifying a Forest

Because backup operations are performed at either the database or the forest level, loading a set of documents into specific forests allows you to effectively perform backup operations on that set of documents (by backing up the database or forest, for example).

Specifying a forest also allows you to have more control over the filesystems in which the documents reside. Each forest configuration includes a directory where the files are stored. By specifying the forest in which a document resides, you can control the directories (and in turn, the filesystems) in which the documents are stored. For example, you might want to place large, frequently accessed documents in a forest which resides on a RAID filesystem with complete failover and redundancy, whereas you might want to place documents which are small and rarely accessed in a forest which resides in a slower (and less expensive) filesystem.

Note: Once a document is loaded into a forest, you cannot move it to another forest. If you want to change the forest in which a document resides, you must reload the document and specify another forest.

5.2.2 Example: Examining a Document to Decide Which Forest to Specify

You can use the `xdmp:document-get` function to a load a document into memory. One use for loading a document into memory is the ability to perform some processing or logic on the document before you load the document onto disk.

For example, if you want to make a decision about which forest to load a document into based on the document contents, you can put some simple logic in your load script as follows:

```
let $memoryDoc := xdmp:document-get("c:\myFiles\newDocument.xml")
let $forest :=
     if( $memoryDoc//ID gt "1000000" )
     then xdmp:forest("LargeID")
     else xdmp:forest("SmallID")
return
     xdmp:document-insert("/myCompany/newDocument.xml",
                          $memoryDoc,
          xdmp:default-permissions(),
          xdmp:default-collections(),
                          0,
                          $forest)
```

This code loads the document `newDocument.xml` into memory, finds the `ID` element in the in-memory document, and then inserts the node into the forest named `LargeID` if the `ID` is greater than 1,000,000, or inserts the node into the forest named `SmallID` if the `ID` is less than 1,000,000.

5.2.3 More Examples

The following command loads the document into the forest named `myForest`:

```
xdmp:document-load("c:\myFile.xml",
       <options xmlns="xdmp:document-load">
          <uri>/myDocs/myDocument.xml</uri>
          <permissions>{xdmp:default-permissions()}</permissions>
          <collections>{xdmp:default-collections()}</collections>
          <repair>full</repair>
          <forests>
            <forest>{xdmp:forest("myForest")}</forest>
          </forests>
        </options> )
```

The following command loads the document into either the forest named `redwood` or the forest named `aspen`:

```
xdmp:document-load("c:\myFile.xml",
       <options xmlns="xdmp:document-load">
          <uri>/myDocs/myDocument.xml</uri>
          <permissions>{xdmp:default-permissions()}</permissions>
          <collections>{xdmp:default-collections()}</collections>
          <repair>full</repair>
          <forests>
            <forest>{xdmp:forest("redwood")}</forest>
            <forest>{xdmp:forest("aspen")}</forest>
          </forests>
        </options> )
```

5.3 Loading External Binaries Using XQuery

You do not use `xdmp:document-load` to load an external binary. To load an external binary document into the database, you first create a binary node using the `xdmp:external-binary` function and then insert the node into the database using `xdmp:document-insert`.

For example, the following code creates a document representing the external binary file `/external/path/sample.jpg`, beginning at offset 1 in the file, with a length of 1M:

```
xdmp:document-insert("/docs/xbin/sample.jpg",
    xdmp:external-binary(
        "/external/path/sample.jpg", 1,1024000))
```

When you provide a length to `xdmp:external-binary`, MarkLogic Server does not verify the existence or size of the external file. If you omit a length when calling `xdmp:external-binary`, the underlying external file must exist, and MarkLogic Server calculates the length in a manner equivalent to calling `xdmp:filesystem-file-length`.

6.0 Loading Content Using the REST API and the Java API

You can use the REST API or the Java API to load content into MarkLogic Server. Using these interfaces, you do not need to write or understand any XQuery. For details, see the *REST Application Developer's Guide* and the *Java Application Developer's Guide*.

7.0 Using MarkLogic Content Pump

MarkLogic Content Pump (mlcp) is a command line tool for getting data into and out of a MarkLogic Server database. This chapter covers the following topics:

- MarkLogic Content Pump Overview
- Terms and Definitions
- Modifying the Example Commands for Windows
- Downloading and Installing mlcp
- Security Considerations
- Using mlcp with a Multi-Host MarkLogic Cluster
- Configuring Distributed Mode
- Importing Content Into MarkLogic Server
- Controlling What is Exported or Copied
- Exporting Content from MarkLogic Server
- Copying Content Between Databases
- Command Line Reference

7.1 MarkLogic Content Pump Overview

Using mlcp, you can import documents and metadata to a database, export documents and metadata from a database, or copy documents and metadata from one database to another. For example:

- Import content into a MarkLogic Server database from flat files, compressed ZIP and GZIP files, or mlcp database archives.
- Create from flat files, delimited text files, Hadoop sequence files, and aggregate XML files. For details, see “Importing Content Into MarkLogic Server” on page 35.
- Import mixed content types from a directory, using the file suffix and MIME type mappings to determine document type. Unrecognized/missing suffixes are imported as binary documents. For details, see “How mlcp Determines Document Type” on page 41.
- Export the contents of a MarkLogic Server database to flat files, a compressed ZIP file, or an mlcp database archive. For details, see “Exporting Content from MarkLogic Server” on page 58.
- Copy content and metadata from one MarkLogic Server database to another. For details, see “Copying Content Between Databases” on page 61.

The mlcp tool has two modes of operation:

- Local: mlcp drives all its work on the host where it is invoked. Resources such as import data and export destination must be reachable from that host.
- Distributed: mlcp distributes its workloads across the nodes in a Hadoop cluster. Resources such as import data and export destination must be reachable from the cluster, which usually means via HDFS.

Distributed mode requires a Hadoop installation. For details, see "Configuring Distributed Mode" on page 34.

To understand the difference between the two modes, consider the following: When loading documents in local mode, all the input data must be reachable from the host on which mlcp is run, and all communication with MarkLogic Server is through that host. Throughput is limited by such things as memory and network bandwidth available to the host running mlcp. When loading documents in distributed mode, multiple nodes in a Hadoop cluster communicate with MarkLogic Server, so greater concurrency can be achieved, while placing fewer resource demands on any one host.

7.2 Terms and Definitions

You should be familiar with the following terms and definitions when using mlcp:

Term	Definition
aggregate	XML content that includes recurring element names and which can be split into multiple documents with the recurring element as the document root. For details, see "Splitting Large XML Files Into Multiple Documents" on page 46.
archive	A compressed MarkLogic Server database archive created using the mlcp export command. You can use an archive to restore or copy database content and metadata with the mlcp import command. For details, see "Exporting to an Archive" on page 60.
HDFS	The Hadoop Distributed File System, which can be used as an input source or an output destination in distributed mode.
sequence file	A flat file of binary key-value pairs in one of the Apache Hadoop `SequenceFile` formats. The mlcp tool only supports importing `Text` and `BytesWritable` values from a sequence file.
split	The unit of work for one thread in local mode or one MapReduce task in distributed mode.

7.3 Modifying the Example Commands for Windows

All the examples in this guide use Unix command line syntax. If you are using mlcp with the Windows command interpreter, `Cmd.exe`, use the following guidelines to construct equivalent commands:

- Replace `mlcp.sh` with `mlcp.bat`.
- For aesthetic reasons, long example command lines are broken into multiple lines using the Unix line continuation character "\". Remove the line continuation characters and place the entire command on one line, or replace the line continuation characters with the Windows equivalent, "^".
- Replace option arguments enclosed in single quotes (') with double quotes ("). If the single-quoted string contains embedded double quotes, escape the inner quotes.
- Escape any unescaped characters that have special meaning to the Windows command interpreter.

For example, the following Unix command line:

```
$ mlcp.sh import -host localhost -port 8006 -username user \
    -password passwd -input_file_path /space/bill/data -mode local \
    -output_uri_replace "/space,'',/bill/data/,'/will/'" \
    -output_uri_prefix /plays
```

Corresponds to this Windows command line:

```
C:\Example> mlcp.bat import -host localhost -port 8006 -username user ^
    -password passwd -input_file_path C:\space\bill -mode local ^
    -output_uri_replace "C:\space,''" ^
    -output_uri_prefix /plays
```

7.4 Downloading and Installing mlcp

This section covers the following topics:

- Supported Platforms
- Required Software
- Installing mlcp

7.4.1 Supported Platforms

In local mode, mlcp is supported on the same platforms as MarkLogic Server, including 64-bit Linux, Windows, Solaris; and Macintosh OS X.

For details, see Supported Platforms in the *Installation Guide*.

Note: Distributed mode is only supported on 64-bit Linux.

7.4.2 Required Software

The following software is required to use mlcp:

- MarkLogic Server version 5.0-5, or MarkLogic Server version 6.0-1 or later, with an XDBC App Server configured to use the target database. The App Server must not be SSL-enabled.
- Java JRE 1.6. The Oracle/Sun JRE is recommended.

In distributed mode, mlcp also requires a Hadoop MapReduce installation. The following distributions are supported. Though only these distributions are supported, mlcp might work with other distributions based on equivalent versions of Apache Hadoop.

- Cloudera's Distribution Including Apache Hadoop (CDH) version 4.3, MRv1 only

  Note: Apache Hadoop only supports the Oracle/Sun JRE, though other JRE's may work. For details, see http://wiki.apache.org/hadoop/HadoopJavaVersions.

7.4.3 Installing mlcp

Follow these instructions to download and install mlcp.

1. Download MarkLogic Content Pump from developer.marklogic.com.

2. Unpack the mlcp distribution to a location of your choice. This creates a directory named `mlcp-HadoopN-version`, where *N* is the compatible Hadoop major version and *version* is the mlcp version. For example, assuming `/space/marklogic` contains the mlcp zip file, then the following commands install mclp under `/space/marklogic/mlcp-Hadoop2-1.1/`:

   ```
   $ cd /space/marklogic
   $ unzip mlcp-Hadoop2-1.1-bin.zip
   ```

3. Optionally, put the mlcp bin directory on your path. For example:

   ```
   $ export PATH=${PATH}:/space/marklogic/mlcp-Hadoop2-1.1/bin
   ```

4. Put the `java` command on your path. For example:

   ```
   $ export PATH=${PATH}:$JAVA_HOME/bin
   ```

5. If you plan to use mlcp in distributed mode, you must have a Hadoop installation and must configure your environment so mlcp can find your Hadoop installation. For details, see "Configuring Distributed Mode" on page 34.

7.5 Security Considerations

When you use mlcp, you supply the name of a user(s) with which to interact with MarkLogic Server. If the user does not have admin privileges, then the user must have at least the privileges listed in the table below.

Note: Additional privileges may be required. These roles only enable use of MarkLogic Server as a data source or destination. For example, these roles do not grant read or update permissions to the database.

mlcp Command	Privilege	Notes
import	`hadoop-user-write`	Applies to the user name specified with `-username`. It is recommended that you also set `-output_permission` to set the permissions on inserted documents.
export	`hadoop-user-read`	Applies to the user name specified with `-username`.
copy	`hadoop-user-read` (input) `hadoop-user-write` (output)	The `-input_username` user have the `hadoop-user-read` privilege on source MarkLogic Server instance. The `-output_username` user must have the `hadoop-user-write` privilege on destination MarkLogic Server instance.

You cannot use mlcp with an SSL enabled App Server.

7.6 Using mlcp with a Multi-Host MarkLogic Cluster

To optimize performance, mlcp interacts with MarkLogic Server at the forest level. When MarkLogic Server is deployed across a multi-host cluster, the forests of a database can be distributed across multiple nodes, which means mlcp can interact with multiple hosts in your MarkLogic cluster. This interaction imposes some cluster configuration requirements:

- All Forest Hosts Must Be Reachable
- Export and Copy Require In-Forest Evaluation of Queries
- Cluster-Wide XDBC Configuration Requirements

For more information about clustering, see Clustering in MarkLogic Server in the *Scalability, Availability, and Failover Guide*.

7.6.1 All Forest Hosts Must Be Reachable

Mlcp uses the connection information provided on the command line to make initial contact with MarkLogic Server and "discover" the network topology of the target database(s). Thereafter, mlcp interacts directly with the hosts that contain forests in the database.

Therefore, all hosts that contain forests of a database accessed by mlcp must be reachable by hostname. That is, the host name under which a forest host joined the MarkLogic cluster must be resolvable by mlcp and the host must be accessible on the network.

For example, if a forest host joins the MarkLogic cluster as HostA, but is only resolvable as HostA-Alias in the subnet from which you execute mlcp, then mlcp will not be able to communicate with that host.

In local mode, the forest hosts must be reachable from the host where you execute the mlcp command. In distributed mode, the forest hosts must reachable from the worker nodes of your Hadoop cluster.

7.6.2 Export and Copy Require In-Forest Evaluation of Queries

When you export or copy documents, mlcp uses in-forest query evaluation to identify the desired documents in each forest of the source database. Therefore, each host that contains a forest in the source database must be configured to act as both an e-node and a d-node.

That is, each host must be capable of providing both query evaluation and data services. A pure d-node (for example, a host with a very small configured expanded tree cache) is not usable with the mlcp export or copy commands.

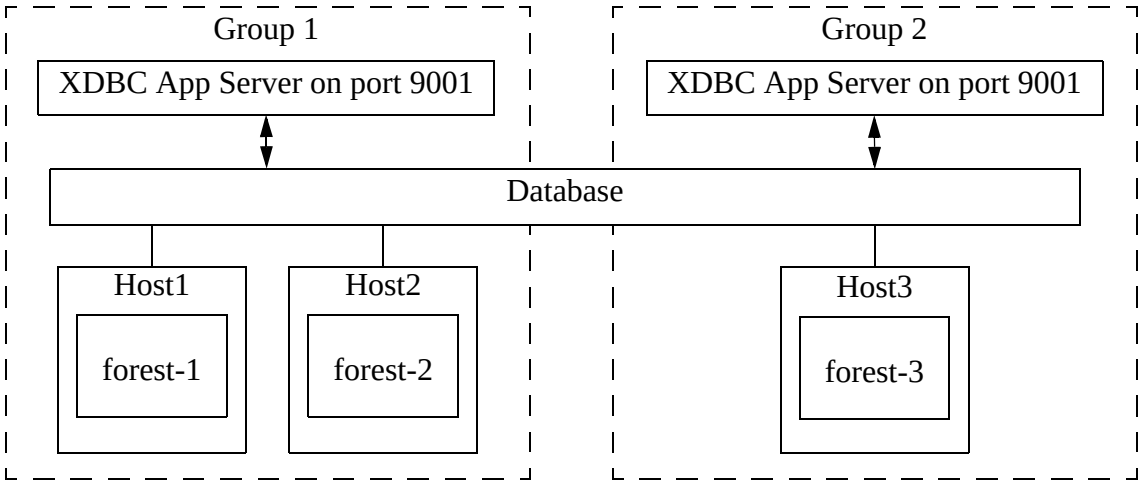
7.6.3 Cluster-Wide XDBC Configuration Requirements

Because mlcp uses an XDBC App Server and in-forest query evaluation, your cluster might need special configuration to support MapReduce jobs.

Each host that has at least one forest attached to a database mlcp interacts with must have an XDBC App Server configured for that database. Additionally, the XDBC App Server must listen on the same port on each host. For a copy operation, the source App Server, database and port can be the same or different from the destination App Server, database and port.

Hosts within a group share the same App Server configuration, so you only need additional App Servers if hosts with forests attached to the input or output database are in multiple groups.

For example, the cluster shown below is properly configured to use the database with mlcp. The database has 3 forests, located on 3 hosts in 2 different groups. Therefore, both Group 1 and Group 2 must make the database accessible on port 9001.



7.7 Configuring Distributed Mode

Using mlcp in distributed mode requires a Hadoop installation. For supported versions, see “Required Software” on page 31.

Hadoop does not have to be installed on the host where you run mlcp, but the Hadoop configuration files must be reachable by mlcp.

- Specifying the Hadoop Configuration File Location
- Setting Custom Hadoop Options and Properties
- Required Hadoop User Privileges

7.7.1 Specifying the Hadoop Configuration File Location

You must tell mlcp where to find the Hadoop configuration files on the host where you run mlcp. Hadoop does not need to be installed on this host, but the Hadoop configuration files must be reachable.

Use one of the following methods to tell mlcp where to find your Hadoop configuration files locally:

- Set the mlcp command line option `-hadoop_conf_dir`. For example:

```
$ mlcp.sh command -hadoop_conf_dir /etc/hadoop/conf
```

- Set the environment variable `HADOOP_CONF_DIR`. For example:

```
$ export HADOOP_CONF_DIR=/etc/hadoop/conf
```

If your Apache Hadoop installation is on a remote host, you can copy the configuration files locally and set `HADOOP_CONF_DIR` (or `-hadoop_conf_dir`) to that directory.

7.7.2 Setting Custom Hadoop Options and Properties

If you want to pass through additional Hadoop properties to tune mlcp's use of Hadoop in distributed mode, use these options:

- `-conf conf_filename` : Pass in a Hadoop configuration properties file.
- `-D property=value` : Pass one Hadoop configuration property setting.

The property names and configuration file syntax is as dictated by Hadoop. For details, consult the documentation for your Hadoop distribution.

> **Note:** These Hadoop options must appear on the command line after `-options_file` (if present) and before any mlcp-specific options.

7.7.3 Required Hadoop User Privileges

When you use distributed mode for import, the user your Hadoop tasks run as must have permission to access the directories or files in specified by `-input_file_path`. Similarly, when you use distributed mode for export, the user must have permission to create directories and files in the directory specified by `-output_file_path`.

7.8 Importing Content Into MarkLogic Server

You can insert content into a MarkLogic Server database from flat files, compressed ZIP and GZIP files, aggregate XML files, Hadoop sequence files, and MarkLogic Server database archives. The input data can be accessed from the native filesystem or HDFS.

For a list of import related options, see "Import Options" on page 64.

This section covers the following topics:

- Supported Input Format Summary
- Understanding Input File Path Resolution
- Controlling Database URIs During Ingestion
- How mlcp Determines Document Type
- Loading Documents from a Directory
- Loading Documents From Compressed Files
- Loading Content and Metadata From an Archive
- Splitting Large XML Files Into Multiple Documents
- Creating Documents from Delimited Text Files
- Creating Documents from Hadoop Sequence Files
- Performance Considerations for Loading Documents

7.8.1 Supported Input Format Summary

The following table provides a quick reference of the supported input file types, along with the allowed document types for each, and whether or not they can be passed to mlcp as compressed files.

-input_file_type	Document Type	-input_compressed permitted
documents	XML, text, or binary; controlled with -document_type.	Yes
archive	As in the database: XML, text, and/or binary documents, plus metadata. The type is not under user control.	No (archives are already in compressed format)
delimited_text	XML	Yes
sequencefile	XML, text or binary; controlled with these options: -input_sequencefile_value_class -input_sequencefile_value_type.	No. However, the contents can be compressed when you create the sequence file. Compression is bound up with the value class you use to generate and import the file.
aggregates	XML	Yes

When the input file type is documents or sequencefile, you must consider the both input format (-input_file_type) and the output document format (-document_type). In addition, for some input formats, input can come from either compressed or uncompressed files (-input_compressed).

Use the -input_file_type option to tell mlcp the format of the data in each input file (or each entry inside a compressed file). This option controls if/how mlcp converts the content into database documents. The default input type is documents, which means each input file or ZIP file entry creates one database document.

All other input file types represent composite input formats which can yield multiple database documents per input file. The mlcp tool supports the following composite input file types: Aggregate XML (aggregate), delimited text files (delimited_text), Hadoop sequence files (sequencefile), and database archives (archive).

The -document_type option controls the database document format when -input_file_type is documents or sequencefile. MarkLogic Server supports text, XML or binary documents. If the document type is not explicitly with these input file types, mlcp uses the input file suffix to determine the type. For details, see “How mlcp Determines Document Type” on page 41.

To illustrate how the `-input_file_type` and `-document_type` fit together, consider a a Hadoop sequence file that contains binary values. You would set the following options:

- `-input_file_type sequencefile`
- `-document_type binary`

If the sequence file contained text rather than binary values, then `-input_file_type` is unchanged, but `-document_type` becomes `text`:

- `-input_file_type sequencefile`
- `-document_type text` (or `xml`, if the values are valid XML)

7.8.2 Understanding Input File Path Resolution

If you do not explicitly include a URI scheme prefix such as `file:` or `hdfs:` on the input file path, mlcp uses the following rules to locate the input path:

- In local mode, mlcp defaults to the local file system (`file`).
- In distributed mode, mlcp defaults to the Hadoop default scheme, which is usually HDFS. The Hadoop default scheme is configurable through the Hadoop configuration parameter `fs.default.name`.

  Note: In distributed mode, the file scheme (`file:`) refers to the local filesystem of the Hadoop cluster nodes to which the job is distributed. For example, if you perform an inport in distributed mode with an input file path that uses the `file:` prefix, the input files must be reachable along that path from all nodes in your Hadoop cluster.

The following example loads files from the local filesystem directory `/space/bill/data`:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8006 -username user \
    -password passwd -input_file_path /space/bill/data -mode local
```

The following example loads files from the native filesystem of each host in a Hadoop cluster, assuming `/space/bill/data` is a shared network path on all hosts in the Hadoop cluster:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8006 -username user \
    -password passwd -input_file_path file:/space/bill/data \
    -mode distributed
```

7.8.3 Controlling Database URIs During Ingestion

By default, the document URIs created by mlcp during ingestion are determined by the input source. The tool supports several command line options for modifying this default behavior.

- Default Document URI Construction
- Transforming the Default URI
- Character Encoding of URIs

7.8.3.1 Default Document URI Construction

The default database URI assigned to ingested documents depends on the input source. Loading content from the local filesystem or HDFS can create different URIs than loading the same content from a ZIP file or archive. Command line options are available for you to modify this behavior.

The following table summarizes the default behavior with several input sources:

Input Source	Default URI	Example
documents in a native or HDFS directory	`/path/filename`	`/space/data/bill/dream.xml`
documents in a ZIP or GZIP file	`/path/inside/zip/file-name`	If the zip file contains a directory entry `bill/`, then the document URI for `dream.xml` in that directory is: `bill/dream.xml`
a GZIP compressed document	`/path/filename-with-out-gzip-suffix`	If the input is `/space/data/big.xml.gz`, the result is `/space/data/big.xml`.
delimited text file	`first_column_value`	For a record of the form “first,second,third” where Column 1 is the id: `first`

Input Source	Default URI	Example
archive	The docment URI from the source database.	
sequence file	The key in a key-value pair	
aggregate	With mlcp v1.0-2 and earlier: *hash_or_taskid-seqnum* Where *hash_or_taskid* is a hash of the split number in local mode and the Hadoop map task id in distributed mode; and *seqnum* begins with 1 and increments for each document created.	1243498756817-1 1243498756817-2
	With mlcp v1.0-3 and later: */path/filename-split_start-seqnum* Where */path/filename* is the full path to the input file, *split_start* is the byte position from the beginning of the split, and *seqnum* begins with 1 and increments for each document created.	For input file /space/data/big.xml: /space/data/big.xml-0-1 /space/data/big.xml-0-2

For example, the following command loads all files from the file systemdirectory `/space/bill/data` into the database attached to the App Server on port 8006. The documents inserted into the database have URIs of form `/space/bill/data/`*filename*.

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8006 -username user \
    -password passwd -input_file_path /space/bill/data -mode local
```

If the `/space/bill/data` directory is zipped up into `bill.zip`, such that `bill/` is the root directory in zip file, then the following command inserts documents with URIs of the form `bill/data/`*filename*:

```
# Windows users, see Modifying the Example Commands for Windows
$ cd /space; zip -r bill.zip bill
$ mlcp.sh import -host localhost -port 8006 -username user \
```

```
    -password passwd -input_file_path /space/bill.zip \
    -mode local -input_compressed true
```

7.8.3.2 Transforming the Default URI

Use the following options to tailor the database URI of inserted documents:

- `-output_uri_replace` performs one or more string substitutions on the default URI.
- `-output_uri_prefix` prepends a string to the URI after substitution.
- `-output_uri_suffix` appends a string to the URI after substitution.

The `-output_uri_replace` option accepts a comma delimited list of regular expression and replacement string pairs. The string portion must be enclosed in single quotes:

```
-output_uri_replace pattern,'string',pattern,'string'
```

For details on the regular expression language supported by `-output_uri_replace`, see “Regular Expression Syntax” on page 76.

Note: These options are applied after the default URI is constructed and encoded, so if the option values contain characters not allowed in a URI, you must encode them yourself. See “Character Encoding of URIs” on page 40.

The following example loads documents from the filesystem directory `/space/bill/data`. The default output URIs would be of the form `/space/bill/data/filename`. The example uses `-output_uri_replace` to replace “bill/data” with “will” and strip off “/space/”, and then adds a “/plays” prefix using `-output_uri_prefix`. The end result is output URIs of the form `/plays/will/filename`.

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8006 -username user \
    -password passwd -input_file_path /space/bill/data -mode local \
    -output_uri_replace "/space,'',/bill/data/,'/will/'" \
    -output_uri_prefix /plays
```

7.8.3.3 Character Encoding of URIs

If a URI constructed by mlcp contains special characters that are not allowed in URIs, mlcp automatically encodes them. This applies to the special characters “ ” (space), “%”, “?” or “#”. For example, “foo bar.xml” becomes “foo%20bar.xml”.

If you supply a URI or URI component, you are responsible for ensuring the result is a legitimate URI. No automatic encoding takes place. This applies to `-output_uri_replace`, `-output_uri_prefix`, and `-output_uri_suffix`. The changes implied by these options are applied after mlcp encodes the default URI.

When mlcp exports documents from the database to the file system (or HDFS) such that the output directory and/or file names are derived from the document URI, the special symbols are decoded. That is, “foo%bar.xml” becomes “foo bar.xml” when exported. For details, see “How URI Decoding Affects Output File Names” on page 58.

7.8.4 How mlcp Determines Document Type

The document type determines what kind of database document mlcp inserts from input content: Text, XML, or binary. Document type is determined in the following ways:

- Document type can be inherent in the input file type. For example, `delimited_text`, `aggregates`, and `rdf` input files always insert XML documents. For details, see “Supported Input Format Summary” on page 36.
- You can specify a document type explicitly with `-document_type`. For example, to load documents as XML, use `-input_file_type documents -document_type xml`. You cannot set an explicit type for all input file types.
- Mlcp can determine document type dynamically from the output document URI and the MarkLogic Server MIME type mappings when you use `-input_file_type documents -document_type mixed`.

If you set `-document_type` to an explicit type such as `-document_type xml`, then mlcp inserts all documents as that type.

If you use `-document_type mixed`, then mlcp determines the document type from the output URI suffix and the MIME type mapping configured into MarkLogic Server. Mixed is the default behavior for `-input_file_type documents`.

> **Note:** You can only use `-document_type mixed` when the input file type is `documents`.

> **Note:** If an unrecognized or unmapped file extension is encountered when loading mixed documents, mlcp creates a binary document.

The following table contains examples of applying the default MIME type mappings to output URIs with various file extensions, an unknown extension, and no extension. The default mapping includes many additional suffixes. You can examine and create MIME type mappings under the Mimetypes section of the Admin Interface. For more information, see “Implicitly Setting the Format Based on the MIME Type” on page 14.

URI	Document Type
/path/doc.xml	XML
/path/doc.jpg	binary
/path/doc.txt	text
/path/doc.unknown	binary
/path/doc-nosuffix	binary

The MIME type mapping is applied to the final output URI. That is, the URI that results from applying the URI transformation options described in “Controlling Database URIs During Ingestion” on page 37. The following table contains examples of how URI transformations can affect the output document type in `mixed` mode, assuming the default MIME type mappings.

Input Filename	URI Options	Output URI	Doc Type
/path/doc.1	None	/path/file.1	binary
/path/doc.1	Add a `.xml` suffix: `-output_uri_suffix ".xml"`	/path/file.xml	XML
/path/doc.1	Replace the unmapped suffix with .txt: `-output_uri_replace "\.\d+,'.txt'"`	/path/file.txt	text

7.8.5 Loading Documents from a Directory

This section discusses importing documents stored as flat files on the native filesystem or HDFS. The following topics are covered:

- Loading a Single File
- Loading All the Files in a Directory
- Filtering Documents Loaded From a Directory

7.8.5.1 Loading a Single File

Use the following procedure to load all the files in a native or HDFS directory and its sub-directories. To load selected files, see “Filtering Documents Loaded From a Directory” on page 44.

1. Set `-input_file_path` to the path to the input file.

2. Set `-input_file_type` if your input files are not documents. For example, if loading from delimited text files, sequence files, aggregate XML files, or database archives.

3. Set `-document_type` if `-input_file_type` is not `documents` and the content type cannot be accurately deduced from the file suffixes as described in “How mlcp Determines Document Type” on page 41.

4. Set `-mode`:

   - If Hadoop is available and you want to distribute the workload across a Hadoop cluster, set `-mode` to `distributed`.
   - If Hadoop is not installed or you want mlcp to perform the work locally, set `-mode` to `local`. (This is the default mode).

   Note: If you are loading from the native filesystem in distributed mode or from HDFS in local mode, you might need to qualify the input file path with a URI scheme of `file:` or `hdfs:`. See “Understanding Input File Path Resolution” on page 37.

By default, the imported document has a database URI based on the input file path. For details, see “Controlling Database URIs During Ingestion” on page 37.

The following example command loads a single XML file:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8006 -username user \
    -password password -input_file_path /space/bill/data/hamlet.xml
```

7.8.5.2 Loading All the Files in a Directory

Use the following procedure to load all the files in a native or HDFS directory and its sub-directories. To load selected files, see “Filtering Documents Loaded From a Directory” on page 44.

1. Set `-input_file_path` to the input directory.

2. Set `-input_file_type` if your input files are not documents. For example, if loading from delimited text files, sequence files, aggregate XML files, or database archives.

3. Set `-document_type` if `-input_file_type` is not `documents` and the content type cannot be accurately deduced from the file suffixes as described in "How mlcp Determines Document Type" on page 41.

4. Set `-mode`:

   - If Hadoop is available and you want to distribute the workload across a Hadoop cluster, set `-mode` to `distributed`.
   - If Hadoop is not installed or you want mlcp to perform the work locally, set `-mode` to `local`. (This is the default mode).

   Note: If you are loading from the native filesystem in distributed mode or from HDFS in local mode, you might need to qualify the input file path with a URI scheme of `file:` or `hdfs:`. See "Understanding Input File Path Resolution" on page 37.

By default, the imported documents have database URIs based on the input file path. For details, see "Controlling Database URIs During Ingestion" on page 37.

The following example command loads all the files in `/space/bill/data`:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8006 -username user \
    -password password -input_file_path /space/bill/data
```

7.8.5.3 Filtering Documents Loaded From a Directory

If -input_file_path names a directory, mlcp loads all the documents in the input directory and subdirectories by default. Use the `-input_file_pattern` option to filter the loaded documents based on a regular expression.

For example, the following command loads only files with a ".xml" suffix from the directory `/space/bill/data`:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8006 -username user \
    -password password -input_file_path /space/bill/data \
    -mode local -input_file_pattern '.*\.xml'
```

The mlcp tool uses Java regular expression syntax. For details, see "Regular Expression Syntax" on page 76.

7.8.6 Loading Documents From Compressed Files

You can load content from one or more compressed files. Filtering of compressed file content is not supported; mlcp loads all documents in a compressed file.

Follow this procedure to load content from one or more ZIP or GZIP compressed files.

1. Set `-input_file_path`:
   - To load from a single file, set `-input_file_path` to the path to the compressed file.
   - To load from multiple files, set `-input_file_path` to a directory containing the compressed files.
2. If the content type cannot be accurately deduced from suffixes of the files inside the compressed file as described in "How mlcp Determines Document Type" on page 41, set `-document_type` appropriately.
3. Set `-input_compressed` to `true`.
4. If the compressed file suffix is not ".zip" or ".gzip", specify the compressed file format by setting `-input_compression_codex` to `zip` or `gzip`.

If you set `-document_type` to anything but `mixed`, then the contents of the compressed file must be homogeneous. For example, all XML or all binary.

The following example command loads binary documents from the compressed file `/space/images.zip` on the local filesystem.

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8006 -username user \
    -password password -mode local -document_type binary \
    -input_file_path /space/images.zip -input_compressed
```

The following example loads all the files in the compressed file `/space/example.jar`, using `-input_compression_codec` to tell mlcp the compression format because of the ".jar" suffix:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8006 -username user \
    -password passwd -mode local -input_file_path /space/example.jar \
    -input_compressed true -input_compression_codec zip
```

If `-input_file_path` is a directory, mlcp loads contents from all compressed files in the input directory, recursing through subdirectories. The input directory must not contain other kinds of files.

By default, the URI prefix on documents loaded from a compressed file mirrors the directory hierarchy inside the compressed file. For example, if a ZIP file contains `bill/data/dream.xml` then the ingested document URI is also `bill/data/dream.xml`. To override this behavior, see "Controlling Database URIs During Ingestion" on page 37.

7.8.7 Loading Content and Metadata From an Archive

Follow this procedure to import content and metadata from a database archive created by the mlcp export command. A database archive is stored in one or more compressed files that contain documents and metadata.

1. Set `-input_file_path`:
   - To load a single archive file, set `-input_file_path` to that file.
   - To load multiple archive files, set `-input_file_path` to a directory containing the compressed archive files.
2. Set `-document_type` to `mixed`, or leave it unset since `mixed` is the default setting.
3. Set `-input_compressed` to `true`.
4. Set `-input_file_type` to `archive`.
5. If the input archive was created without any metadata, set `-archive_metadata_optional` to true. If this is not set, an exception is thrown if the archive contains no metadata.
6. If you want to exclude some or all of the document metadata in the archive:
   - Set `-copy_collections` to `false` to exclude document collections metadata.
   - Set `-copy_permissions` to `false` to exclude document permissions metadata.
   - Set `-copy_properties` to `false` to exclude document properties.
   - Set `-copy_quality` to `false` to exclude document quality metadata.

An archive is assumed to contain metadata. However, it is possible to create archives without metadata by setting all the metadata copying options (`-copy_collections`, `-copy_permissions`, etc.) to false during export. If an archive does not contain metadata, you must set `-archive_metadata_optional` to tell mlcp to proceed in the absence of metadata.

The following example command loads the database archive in `/space/archive_dir`:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8006 -username user \
    -password password -mode local -input_file_type archive \
    -input_file_path /space/archive_dir
```

7.8.8 Splitting Large XML Files Into Multiple Documents

Very large XML files often contain *aggregate* data that can be disaggregated by splitting it into multiple smaller documents rooted at a recurring element. Disaggregating large XML files consumes fewer resources during loading and improves performance when searching and retrieving content.

The following mlcp options support creating multiple documents from aggregate data:

- `-aggregate_record_element`
- `-aggregate_uri_id`
- `-aggregate_record_namespace`

You can disaggregate XML when loading from either flat or compressed files. For more information about working with compressed files, see “Loading Documents From Compressed Files” on page 44.

Follow this procedure to create documents from aggregate XML input:

1. Set `-input_file_path`:
   - To load from a single file, set `-input_file_path` to the path to the aggregate XML file.
   - To load from multiple files, set `-input_file_path` to a directory containing the aggregate files. The directory must not contain other kinds of files.
2. If you are loading from a compressed file, set `-input_compressed`.
3. Set `-input_file_type` to `aggregates`.
4. Set `-aggregate_record_element` to the element QName of the node to use as the root for all inserted documents. See the example below. The default is the first child element under the root element.

   Note: The element QName should appear at only one level. You cannot specify the element name using a path, so disaggregation occurs everywhere that name is found.
5. Optionally, override the default document URI by setting `-aggregate_uri_id` to the name of the element from which to derive the document URI.

   The default URI is *`hashcode-seqnum`* in local mode and *`taskid-seqnum`* in distributed mode. If there are multiple matching elements, the first match is used.
6. If the aggregate record element is in a namespace, set `-aggregate_record_namespace` to the input namespace.

The example below uses the following input data:

```
$ cat > example.xml
<?xml version="1.0" encoding="UTF-8"?>
<people>
  <person>
    <first>George</first>
    <last>Washington</last>
  </person>
  <person>
    <first>Betsy</first>
    <last>Ross</last>
  </person>
</people>
```

The following command breaks the input data into a document for each `<person>` element. The `-aggregate_uri_id` and other URI options give the inserted documents meaningful names. The command creates URIs of the form "/people/lastname.xml" by using the `<last/>` element as the aggregate URI id, along with an output prefix and suffix:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8006 -username user \
    -password password -mode local -input_file_path example.xml \
    -input_file_type aggregates -aggregate_record_element person \
    -aggregate_uri_id last -output_uri_prefix /people/ \
    -output_uri_suffix .xml
```

The command creates two documents: `/people/Washington.xml` and `/people/Ross.xml`. For example, `/people/Washington.xml` contains:

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <first>George</first>
    <last>Washington</last>
</person>
```

If the input data is in a namespace, set `-aggregate_record_namespace` to that namespace. For example, if the input data is modified to include a namespace:

```
$ cat > example.xml
<?xml version="1.0" encoding="UTF-8"?>
<people xmlns="http://marklogic.com/examples">...</people>
```

Then mlcp ingests no documents unless you set `-aggregate_record_namespace`. Setting the namespace creates two documents in the namespace "http://marklogic.com/examples". For example, after running the following command:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8006 -username user \
    -password password -mode local -input_file_path example.xml \
    -input_file_type aggregates -aggregate_record_element person \
    -aggregate_uri_id last -output_uri_prefix /people/ \
    -output_uri_suffix .xml \
    -aggregate_record_namespace "http://marklogic.com/examples"
```

The document with URI "/people/Washington.xml" contains :

```
<?xml version="1.0" encoding="UTF-8"?>
<person xmlns="http://marklogic.com/examples">
    <first>George</first>
    <last>Washington</last>
</person>
```

7.8.9 Creating Documents from Delimited Text Files

Use the `delimited_text` input file type to import content from a delimited text file and create an XML document corresponding to each line.

The first line in the input file should contain column headers. For each line after the header line, mlcp creates an XML document with a root node of `<root>` and child elements with names corresponding to each column title.

For example, given the following data and mlcp command:

```
# Windows users, see Modifying the Example Commands for Windows
$ cat example.csv
first,last
george,washington
betsy,ross

$ mlcp.sh ... -mode local -input_file_path /space/mlcp/data \
    -input_file_type delimited_text
```

mlcp creates two documents with the following contents:

```
<root>
  <first>george</first>
  <last>washington</last>
</root>

<root>
  <first>betsy</first>
  <last>ross</last>
</root>
```

By default, the document URIs use the value in the first column. In the example above, the two documents have URIs corresponding to the “first” value, or “george” and “betsy”. Use `-delimited_uri_id` to choose a different column. For example, the following command creates the documents “washington” and “ross”:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh ... -mode local -input_file_path /space/mlcp/data \
    -input_file_type delimited_text -delimited_uri_id last
```

You can further tailor the URIs using `-output_uri_prefix` and `-output_uri_suffix`. For details, see “Controlling Database URIs During Ingestion” on page 37.

7.8.10 Creating Documents from Hadoop Sequence Files

A Hadoop sequence file is a flat binary file of key-value pairs. You can use mlcp to create a document from each key-value pair. The only supported value types are `Text` and `BytesWritable`. This section covers the following topics:

- Basic Steps
- Implementing the Key and Value Interfaces
- Deploying your Key and Value Implementation
- Loading Documents From Your Sequence Files
- Running the SequenceFile Example

7.8.10.1 Basic Steps

You must implement a Hadoop `SequenceFile` reader and writer that also implements 2 special mlcp interfaces. To learn more about Apache Hadoop `SequenceFile`, see http://wiki.apache.org/hadoop/SequenceFile/.

1. Implement `com.marklogic.contentpump.SequenceFileKey` and `com.marklogic.contentpump.SequenceFileValue`.

2. Generate one or more sequence files using your classes.

3. Deploy your classes into *`mlcp_install_dir`*`/lib`.

4. Use the mlcp import command to create documents from your sequence files.

The source distribution of mlcp, available from http://developer.marklogic.com, includes an example in `com.marklogic.contentpump.examples`.

7.8.10.2 Implementing the Key and Value Interfaces

You must read and write your sequence files using classes that implement `com.marklogic.contentpump.SequenceFileKey` and `com.marklogic.contentpump.SequenceFileValue`. These interfaces are included in the mlcp jar file:

```
mlcp_install_dir/lib/mlcp-HadoopN-version.jar
```

Where *N* is the Hadoop major version supported by your mlcp installation and *`version`* is your mlcp version. For example, if you install mlcp v1.1 compatible with Hadoop v2 to `/opt/mlcp-Hadoop2-1.1`, then the jar file is:

```
/opt/mlcp-Hadoop2-1.1/lib/mlcp-Hadoop2-1.1.jar
```

Source and an example implementation are available in the mlcp source distribution on developer.marklogic.com.

Your key class must implement the following interface:

```
package com.marklogic.contentpump;
```

```
import com.marklogic.mapreduce.DocumentURI;

public interface SequenceFileKey {
    DocumentURI getDocumentURI();
}
```

Your value class must implement the following interface:

```
package com.marklogic.contentpump;

public interface SequenceFileValue<T> {
    T getValue();
}
```

For an example, see `com.marklogic.contentpump.example.SimpleSequenceFileKey` and `com.marklogic.contentpump.example.SimpleSequenceFileValue`.

These interfaces depend on Hadoop and the MarkLogic Connector for Hadoop. The connector library is included in the mlcp distribution as:

mlcp_install_dir/lib/marklogic-mapreduce*N-version*.jar

where *N* is the Hadoop major version and *version* is the connector version. The Hadoop major version will correspond to the Hadoop major version of your mlcp distribution. For example, if you install the Hadoop v2 compatible version of mlcp, then the connector jar file name might be:

```
marklogic-mapreduce2-1.2.jar
```

For details, see the *MarkLogic Connector for Hadoop Developer's Guide* and the *MarkLogic Hadoop MapReduce Connector API*.

You must implement a sequence file creator. You can find an example in `com.marklogic.contentpump.examples.SimpleSequenceFileCreator`.

When compiling your classes, include the following on the Java class path:

- *mlcp_install_dir*/lib/mlcp-Hadoop*N-version*.jar
- *mlcp_install_dir*/lib/marklogic-mapreduce*N-version*.jar
- *mlcp_install_dir*/lib/hadoop-core-*version*.jar

For example, if you are using the Hadoop v2 compatible version of mlcp:

```
$ javac -cp
$MLCP_DIR/lib/mlcp-Hadoop2-1.1.jar:$MLCP_DIR/lib/marklogic-mapreduce2-
1.2.jar:$MLCP_DIR/lib/hadoop-core-2.0.0-mr1-cdh4.3.0.jar MyKey.java
MyValue.java

$ jar -cf myseqfile.jar *.class
```

7.8.10.3 Deploying your Key and Value Implementation

Once you compile your `SequenceFileKey` and `SequenceFileValue` implementations into a JAR file, copy your JAR file and any dependent libraries into the mlcp `lib/` directory so that mlcp can find your classes at runtime. For example:

```
$ cp myseqfile.jar /space/mlcp-Hadoop2-1.1/lib
```

7.8.10.4 Loading Documents From Your Sequence Files

Once you have created one or more sequence files using your implementation, you can create a document from each key-value pair using the following procedure:

1. Set `-input_file_path`:
   - To load from a single file, set `-input_file_path` to the path to the file.
   - To load from multiple files, set `-input_file_path` to a directory containing the sequence files.
2. Set `-sequencefile_key_class` to the name of your `SequenceFileKey` implementation.
3. Set `-sequencefile_value_class` to the name of your `SequeneFileValue` implementation.
4. Set `-sequencefile_value_type` to either `Text` or `BytesWritable`, depending on the contents of your sequence files.
5. Set `-input_file_type` to `sequencefile`.

By default, the key in each key-value pair is used as the document URI. You can further tailor the URI using command line options, as described in “Controlling Database URIs During Ingestion” on page 37.

For an example, see “Running the SequenceFile Example” on page 52.

7.8.10.5 Running the SequenceFile Example

This section walks you through creating a sequence file and loading its contents as documents.

Create an input text file from which to create a sequence file. The file should contain pairs of lines where the first line is a URI that acts as the key, and the second line is the value. For example:

```
$ cat > seq_input.txt
/doc/foo.xml
<foo/>
/doc/bar.xml
<bar/>
```

To use the example classes provided with mlcp, put the following libraries on your Java classpath:

- *mlcp_install_dir*/lib/mlcp-Hadoop*N-version*.jar
- *mlcp_install_dir*/lib/hadoop-core-*version*.jar
- *mlcp_install_dir*/lib/commons-logging-1.1.1.jar
- *mlcp_install_dir*/lib/marklogic-mapreduce*N-version*.jar

For example, if you install the CDH 4.3 MRv1 compatible version of mlcp v1.1, then put the following libraries on your Java classpath:

- *mlcp_install_dir*/lib/mlcp-Hadoop2-1.1.jar
- *mlcp_install_dir*/lib/hadoop-core-2.0.0-mr1-cdh4.3.0.jar
- *mlcp_install_dir*/lib/commons-logging-1.1.1.jar
- *mlcp_install_dir*/lib/marklogic-mapreduce2-1.2.jar

Generate a sequence file from your test data using `com.marklogic.contentpump.examples.SimpleSequenceFileCreator`. The first argument to program is the output sequence file name. The second argument is the input data file name. The following command generates `seq_output` from `seq_input.txt`.

```
$ java com.marklogic.contentpump.examples.SimpleSequenceFileCreator
seq_output seq_input.txt
```

Load the contents of the sequence file into MarkLogic Server:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -user user -password password -host localhost \
    -port 8006 -input_file_path seq_output -mode local \
    -input_file_type sequencefile -sequencefile_key_class \
    com.marklogic.contentpump.examples.SimpleSequenceFileKey \
    -sequencefile_value_class \
    com.marklogic.contentpump.examples.SimpleSequenceFileValue \
    -sequencefile_value_type Text -document_type xml
```

Two documents are created in the database with URIs `/doc/foo.xml` and `/doc/bar.xml`.

7.8.11 Performance Considerations for Loading Documents

MarkLogic Content Pump comes configured with defaults that should provide good performance under most circumstances. This section presents some performance tradeoffs to consider if you want to try to optimize throughput for your workload.

This section covers the following topics:

- Time vs. Space: Configuring Batch and Transaction Size
- Time vs. Correctness: Using Direct Forest Updates
- Tuning Split Size for Local Mode

- Tuning Split Size for Distributed Mode
- Reducing Memory Consumption With Streaming

7.8.11.1 Time vs. Space: Configuring Batch and Transaction Size

You can tune the document insertion throughput and memory requirements of your job by configuring the batch size and transaction size of the job.

- `-batch_size` controls the number of updates per request to the server.
- `-transaction_size` controls the number of requests to the server per transaction.

The default batch size is 100. The default transaction size is 10. This means that the default maximum number of updates per transaction is 1000.

Selecting a batch size is a speed vs. memory tradeoff. Each request to the server introduces overhead because extra work must be done. However, unless you use `-streaming` or `-document_type mixed`, all the updates in a batch stay in memory until a request is sent, so larger batches consume more more memory.

Transactions introduce overhead on MarkLogic Server, so performing multiple updates per transaction can improve insertion throughput. However, an open transaction holds locks on fragments with pending updates, potentially increasing lock contention and affecting overall application performance.

It is also possible to overwhelm MarkLogic Server if you have too many concurrent sessions active.

7.8.11.2 Time vs. Correctness: Using Direct Forest Updates

The mlcp import and copy commands perform best when documents can be loaded directly into the destination forests, without going through an e-node. However, direct forest updates can create duplicate URIs under the following circumstances:

- Content with the same URI already exists in the database, and
- The content was inserted using user-specified forest placement, or the number of forests available for updates changes after document creation.

User-specified forest placement occurs when you specify an explicit forest id during document loading, such as the `$forest-ids` parameter to `xdmp:document-load`.

To prevent duplicate URIs, mlcp defaults to a slower protocol when loading documents. You can override this behavior using the `-fastload` or `-output_directory` options. (Using `-output_directory` implies `-fastload`). You can safely enable `-fastload` if the number of forests in the database will not change while mlcp is loading, and at least one of the following is true:

- The mlcp job only creates new documents. That is, you are certain that the URIs are not in use by any document or property fragments already in the database.
- The URIs may already be in use, but all these conditions are true:
    - The in-use URIs were not originally inserted using user-specified forest placement.
    - The number of forests available for updates in the database has not changed since the documents were created.
    - The order of forest assignments has not changed since the documents were created. Forest assignment order can change when de-assigning and then re-assigning forests to a database.

    Note: If you import data into database hosted by MarkLogic 7 or later, the database you import data into with `-fastload` or `-output_directory` must be using the legacy assignment policy. For details, refer to the documentation for your current version of MarkLogic Server.

Changing the number of forests available for updates while a job is running can also cause duplicate URIs or insertion failures. The number of forests available for update can change for several reasons, including the following:

- Adding a new forest
- Removing an existing forest
- Changing the updates-allowed state of forest. For example, calling `admin:forest-set-updates-allowed`.

7.8.11.3 Tuning Split Size for Local Mode

This discussion applies only to importing whole documents from flat or compressed files in local mode. It does not apply to creating documents from composite files such as delimited text files, sequence files, or aggregate XML files. Split size for composite files is not tunable.

In local mode, a split defines the unit of work per thread. The ideal split size is one that keeps all your threads busy. The default split size is 32M for local mode, and the HDFS block size for distributed mode.

For example, suppose `-thread_count` is 10 and `-max_split_size` is 32M and your content consists of 120 small documents 1M in length. Then 32 documents fit into each split and there will only be 4 splits. The run uses only 4 of 10 available threads, leaving the other 6 idle. If you tune `-max_split_size` down to 12M, you get 10 splits and can maximize concurrency.

7.8.11.4 Tuning Split Size for Distributed Mode

The following discussion applies only to importing whole documents from flat or compressed files in distributed mode. It does not apply to creating documents from composite files such as delimited text files, sequence files, and aggregate XML files. Split size for composite files is not tunable.

Distributed mode uses Hadoop to import documents from multiple tasks, running on the nodes of a Hadoop cluster. A *split* is a unit of work for one Hadoop task.

Tuning Hadoop performance, including split size, is a complex topic outside the scope of this document. However, for best performance, tune split size to maximize parallelism across your Hadoop cluster, with each task taking at least a couple minutes. If your split size is too small and each task only runs for a very short time, the task overhead can degrade overall performance.

In distributed mode, split size is determined by the following formula:

```
max(min_split_size, min(max_split_size, block_size))
```

The default `min_split_size` is 0 and the default `max_split_size` is `Long.MAX` (the maximum signed long integer). The `block_size` depends on your Hadoop configuration, but the default is 64M. You can configure the min and max split size using the mlcp options `-min_split_size` and `-max_split_size`. You can only tune block size through Hadoop configuration settings.

In addition to balancing the workload across MapReduce tasks, you must also consider the load on MarkLogic Server. Too many concurrent sessions can over tax CPU and memory resources.

7.8.11.5 Reducing Memory Consumption With Streaming

The streaming protocol allows you to insert a large document into the database without holding the entire document in memory. Streaming uploads documents to MarkLogic Server in 128k chunks.

Streaming content into the database usually requires less memory on the host running mlcp, but ingestion can be slower because it introduces additional network overhead. Streaming also does not take advantage of mlcp's builtin retry mechanism. If an error occurs that is normally retryable, the job will fail.

> **Note:** Streaming is only usable when `-input_file_type` is `documents`. You cannot use streaming with delimited text files, sequence files, or archives.

To use streaming, enable the `-streaming` option. For example:

```
# Windows users, see Modifying the Example Commands for Windows
$mlcp.sh import -username user -password password -host localhost \
    -port 8006 -input_file_path /my/dir -streaming
```

7.9 Controlling What is Exported or Copied

By default, mlcp exports all documents or all documents and metadata in the database, depending on whether you are exporting in document or archive format or copying the database. Several command line options are available to enable customization. This section covers the following topics:

- Filtering Document Exports
- Filtering Archive and Copy Contents

7.9.1 Filtering Document Exports

This section covers options available for filtering what is exported by the mlcp export command when `-output_type` is `document`.

By default, mlcp exports all documents in the database. That is, mlcp exports the equivalent of `fn:collection()`. The following options allow you to filter what is exported:

- `-directory_filter` - export only the documents in the listed database directories. You cannot use this option with `-collection_filter`.
- `-collection_filter` - export only the documents in the listed collections. You cannot use this option with `-directory_filter`.

7.9.2 Filtering Archive and Copy Contents

This section covers options available for controlling what is exported by mlcp export when `-output_type` is `archive`, or what is copied by mlcp copy.

By default, all documents and metadata are exported/copied. The following options allow you to modify this behavior:

- `-directory_filter` - export/copy only the documents in the listed database directories, including related metadata. You cannot use this option with `-collection_filter`.
- `-collection_filter` - export/copy only the documents in the listed collections, including related metadata. You cannot use this options with `-directory_filter`.
- `-copy_collections` - whether to include collection metadata
- `-copy_permissions` - whether to include permissions metadata
- `-copy_properties` - whether to include naked and document properties
- `-copy_quality` - whether to include document quality metadata

If you set all the `-copy_*` options to `false` when exporting to an archive, the archive contains no metadata. When you import an archive with no metadata, you must set `-archive_metadata_optional` to `true`.

7.10 Exporting Content from MarkLogic Server

You can export content in a MarkLogic Server database to files or an archive. Use archives to copy content from one MarkLogic Server database to another. Output can be written to the native filesystem or to HDFS.

For a list of export related command line options, see “Export Options” on page 70.

This section covers the following topics:

- Potential Export Output Inconsistency
- How URI Decoding Affects Output File Names
- Exporting Documents as Files
- Exporting Documents to a Compressed File
- Exporting to an Archive

7.10.1 Potential Export Output Inconsistency

If consistency between export output and the database contents is important, ensure there are no updates happening in the database during the export. Consistency is not guaranteed if there are updates to the database while the export is running.

For example, if a document changes after mlcp has exported a copy of the document, then the export output does will not match the database state. If the document changes after export begins but before the document is copied out, the new version will be exported, not the version that was present when mlcp began running.

7.10.2 How URI Decoding Affects Output File Names

This discussion only applies when `-output_type` is `document`.

When you export a document to a file (or to a file in a compressed file), the output file name is based on the document URI. The document URI is decoded to form the file name. For example, if the document URI is “foo%20bar.xml”, then the output file name is “foo bar.xml”.

If the document URI does not conform to the standard URI syntax of RFC 3986, decoding may fail, resulting in unexpected file names. For example, if the document URI contains unescaped special characters then the raw URI may be used.

If the document URI contains a scheme, the scheme is removed. If the URI contains both a scheme and an authority, both are removed. For example, if the document URI is “file:foo/bar.xml”, then the output file path is *`output_file_path`*`/foo/bar.xml`. If the document URI is “http://marklogic.com/examples/bar.xml” (contains a scheme and an authority), then the output file path is *`output_file_path`*`/examples/bar.xml`.

If the document URI includes directory steps, then corresponding output subdirectories are created. For example, if the document URI is "/foo/bar.xml", then the output file path is *output_file_path*/foo/bar.xml.

7.10.3 Exporting Documents as Files

Use the mlcp `export` command to export documents in their original format as files on the native filesystem or HDFS. For example, you can export an XML document as a text file containing XML, or a binary document as a JPG image.

To export documents from a database as files:

1. Select the files to export. For details, see "Filtering Document Exports" on page 57.
    - To select documents in one or more collections, set `-collection_filter` to a comma separated list of collection URIs.
    - To select documents in one or more database directories, set `-directory_filter` to a comma separated list of directory URIs.
    - To select all documents in the database, leave `-collection_filter` and `-directory_filter` unset.
2. Set `-output_file_path` to the destination file or directory on the native filesystem or HDFS.
3. To prettyprint exported XML when using local mode, set `-indented` to true.

Directory names specified with -directory_filter should end with "/".

> **Note:** Document URIs are URI-decoded before filesystem directories or filenames are constructed for them. For details, see "How URI Decoding Affects Output File Names" on page 58.

For a full list of export options, see "Export Options" on page 70.

The following example exports selected documents in the database to the native filesystem directory `/space/mlcp/export/files`. The directory filter selects only the documents in /plays.

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh export -host localhost -port 8006 -username user \
    -password password -mode local -output_file_path \
    /space/mlcp/export/files -directory_filter /plays/
```

7.10.4 Exporting Documents to a Compressed File

Use the mlcp `export` command to export documents in their original format as files in a compressed ZIP file on the native filesystem or HDFS.

To export documents from a database as files:

1. Select the files to export. For details, see "Filtering Document Exports" on page 57.

    - To select documents in one or more collections, set `-collection_filter` to a comma separated list of collection URIs.
    - To select documents in one or more database directories, set `-directory_filter` to a comma separated list of directory URIs.
    - To select all documents in the database, leave `-collection_filter` and `-directory_filter` unset.

2. Set `-output_file_path` to the destination directory on the native filesystem or HDFS. This directory must not already exist.

3. Set `-compress` to `true`.

4. To prettyprint exported XML when using local mode, set `-indented` to true.

For a full list of export options, see "Export Options" on page 70.

The zip files created by export have filenames of the form *`timestamp-seqnum`*`.zip`.

The following example exports all the documents in the database to the directory `/space/examples/export` on the native filesystem.

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh export -host localhost -port 8006 -username user \
    -password password -mode local \
    -output_file_path /space/examples/export -compress true

$ ls /space/examples/export
20120823135307-0700-000000-XML.zip
```

7.10.5 Exporting to an Archive

Use the mlcp export command with an output type of `archive` to create a database archive that includes content and metadata. You can use the mlcp import command to copy the archive to another database or restore database contents.

To export database content to an archive file with mlcp:

1. Select the documents to export. For details, see "Filtering Archive and Copy Contents" on page 57.

    - To select documents in one or more collections, set `-collection_filter` to a comma separated list of collection URIs.

- To select documents in one or more database directories, set `-directory_filter` to a comma separated list of directory URIs.
- To select all documents in the database, leave `-collection_filter` and `-directory_filter` unset.

2. Set `-output_file_path` to the destination directory on the native filesystem or HDFS. This directory must not already exist.

3. Set `-output_type` to `archive`.

4. If you want to exclude some or all document metadata from the archive:

   - Set `-copy_collections` to `false` to exclude document collections metadata.
   - Set `-copy_permissions` to `false` to exclude document permissions metadata.
   - Set `-copy_properties` to `false` to exclude document properties.
   - Set `-copy_quality` to `false` to exclude document quality metadata.

For a full list of export options, see "Export Options" on page 70.

The following example exports all documents and metadata to the directory `/space/examples/exported`. After export, the directory contains one or more compressed archive files.

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh export -host localhost -port 8006 -username user \
    -password password -mode local \
    -output_file_path /space/examples/exported -output_type archive
```

The following example exports only documents in the database directory `/plays/`, including their collections, properties, and quality, but excluding permissions:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh export -host localhost -port 8006 -username user \
    -password password -mode local \
    -output_file_path /space/examples/exported -output_type archive \
    -copy_permissions false -directory_filter /plays/
```

You can use the mlcp import command to import an archive into a database. For details, see "Loading Content and Metadata From an Archive" on page 45.

7.11 Copying Content Between Databases

Use the mlcp copy command to copy content and associated metadata from one MarkLogic Server database to another when both are reachable on the network.

To copy one database to another with mclp:

1. Set `-input_host`, `-input_port`, `-input_username`, and `-input_password` to identify the source MarkLogic Server instance and user.

2. Set `-output_host`, `-output_port`, `-output_username`, and `-output_password` to identify the destination MarkLogic Server instance and user.

3. Select what documents to copy. For details, see "Filtering Archive and Copy Contents" on page 57.

    - To select documents in one or more collections, set `-collection_filter` to a comma separated list of collection URIs.
    - To select documents in one or more database directories, set `-directory_filter` to a comma separated list of directory URIs.
    - To select all documents in the database, leave `-collection_filter` and `-directory_filter` unset.

4. If you want to exclude some or all source document metadata:

    - Set `-copy_collections` to `false` to exclude document collections metadata.
    - Set `-copy_permissions` to `false` to exclude document permissions metadata.
    - Set `-copy_properties` to `false` to exclude document properties.
    - Set `-copy_quality` to `false` to exclude document quality metadata.

5. If you want to add or override document metadata in the destination database:

    - Set `-output_collections` to add destination documents to a collection.
    - Set `-output_permissions` to add permissions to destination documents.
    - Set `-output_quality` to set the quality of destination documents.

6. If you want the destination documents to have database URIs different from the source URIs, set `-output_uri_replace`, `-output_uri_prefix`, and/or `-output_uri_suffix`. For details, see "Controlling Database URIs During Ingestion" on page 37.

For a complete list of mlcp copy command options, see "Copy Options" on page 73.

The following example copies all documents and their metadata from the source database to the destination database:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh copy -mode local -input_host srchost -input_port 8006 \
    -input_username user1 -input_password password1 \
    -output_host desthost -output_port 8010 -output_username user2 \
    -output_password password2
```

The following example copies selected documents, excluding the source permissions and adding the documents to 2 new collections in the destination database:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh copy -mode local -input_host srchost -input_port 8006 \
    -input_username user1 -input_password password1 \
    -output_host desthost -output_port 8010 -output_username user2 \
    -output_password password2 -copy_permissions false \
    -output_collections shakespeare,plays
```

7.12 Command Line Reference

This section covers the following topics:

- Command Line Summary
- Setting Java Virtual Machine (JVM) Options
- Import Options
- Export Options
- Copy Options
- Regular Expression Syntax
- Options File Syntax

7.12.1 Command Line Summary

The mlcp command line has the following structure:

- Linux, Solaris, and OS X: `mlcp.sh` *`command options`*
- Windows: `mlcp.bat` *`command options`*

Where *`command`* is one of the commands in the table below. Each command has a set of command-specific options.

Command	Description
import	Import data from the file system, the Hadoop Distributed File System (HDFS), or standard input to a MarkLogic Server database. See “Import Options” on page 64.
export	Export data from a MarkLogic Server database to the file system or HDFS. See “Export Options” on page 70.
copy	Copy data from one MarkLogic Server database to another. See “Copy Options” on page 73.
help	Display brief help about mlcp. To get a list command line options for a particular command, use `mlcp.sh command -help`.

In addition to the command-specific options, mlcp enables you to pass additional settings to Hadoop MapReduce when using `-mode distributed`. This feature is for advanced users who are familiar with MapReduce. For details, see "Setting Custom Hadoop Options and Properties" on page 35.

> **Note:** If you use Hadoop-specific options such as `-conf` or `-D`, they must appear after `-options_file` (if present) and before any mlcp-specific options.

Options can also be specified in an options file using `-options_file`. Options files and command line options can be used together. For details, see "Options File Syntax" on page 77.

Note the following conventions for command line options to mlcp:

- Prefix options with a single dash (-).
- Option names are case-sensitive.
- If an option has a value, separate the option name and value with whitespace. For example: `mlcp import -username admin`
- If an option has a predefined set of possible values, such as `-mode`, the option values are case-insensitive unless otherwise noted.
- If an option appears more than once on the command line, the first occurrence is used.
- When string option values require quoting, use single quotes. For example: `-output_uri_replace "this,'that '"`.
- The value of a boolean typed option can be omitted. If the value is omitted, true is implied. For example, `-copy_collections` is equivalent to `-copy_collections true`.

7.12.2 Setting Java Virtual Machine (JVM) Options

The mlcp tool is a Java application. You can pass extra parameters to the JVM during an mlcp command using the environment variable `JVM_OPTS`.

For example, the following command passes the setting "-Xmx100M" to the JVM to increase the JVM heap size for a single mclp run:

```
$ JVM_OPTS='-Xmx100M' mclp.sh import ...
```

7.12.3 Import Options

This section summarizes the command line options available with the mlcp `import` command. For examples and use cases, see "Importing Content Into MarkLogic Server" on page 35.

The following command line options are required when using the `import` command:

Option	Description
-host *string*	Hostname of the destination MarkLogic Server.
-port *number*	Port number of the destination MarkLogic Server. There should be an XDBC App Server on this port, listening on behalf of the target database. The App Server must not be SSL-enabled.
-username *string*	MarkLogic Server user with which to import documents.
-password *string*	Password for the MarkLogic Server user specified with `-username`.

The following table lists additional command line options:

Option	Description
-aggregate_record_element *string*	When splitting an aggregate input file into multiple documents, the name of the element to use as the output document root. Default: The first child element under the root element.
-aggregate_record_namespace *string*	The namespace of the element specificed by `-aggregate_record_element_name`. Default: No namespace.
-aggregate_uri_id *string*	When splitting an aggregate input file into multiple documents, the element or attribute name within the document root to use as the document URI. Default: In local mode, *`hashcode-seqnum`*, where the hashcode is derived from the split number; in distribute mode, *`taskid-seqnum`*.
-batch_size *number*	The number of documents to process in a single request to MarkLogic Server. Default: 100.
-conf *filename*	Pass extra setting to Hadoop when using distributed mode. For details, see “Setting Custom Hadoop Options and Properties” on page 35. This option must appear before mlcp-specific options.

Option	Description
-content_encoding *string*	The character encoding of input documents when `-input_file_type` is `documents`, `aggregates`, or `delimited_text`. The option value must be a character set name accepted by your JVM; see `java.nio.charset.Charset`. Default: The platform default encoding for the host on which mlcp runs.
-copy_collections *boolean*	When importing documents from an archive, whether to copy document collections from the source archive to the destination. Only applies with `-input_file_type archive`. Default: true.
-copy_permissions *boolean*	When importing documents from an archive, whether to copy document permissions from the source archive to the destination. Only applies with `-input_file_type archive`. Default: true.
-copy_properties *boolean*	When importing documents from an archive, whether to copy document properties from the source archive to the destination. Only applies with `-input_file_type archive`. Default: true.
-copy_quality *boolean*	When importing documents from an archive, whether to copy document quality from the source archive to the destination. Only applies with `-input_file_type archive`. Default: true.
-D *property=value*	Pass a configuration property setting to Hadoop when using distributed mode. For details, see "Setting Custom Hadoop Options and Properties" on page 35. This option must appear before mlcp-specific options.
-delimiter *character*	When importing content with `-input_file_type delimited_text`, the delimiting character. Default: comma (,).
-delimited_uri_id *string*	When importing content `-input_file_type delimited_text`, the column name that contributes to the id portion of the URI for inserted documents. Default:

Option	Description
-document_type *string*	The type of document to create when `-input_file_type` is `documents` or `sequencefile`. Accepted values: `mixed` (`documents` only), `xml`, `text`, `binary`. Default: `mixed` for `documents`, xml for `sequencefile`.
-fastload *boolean*	Whether or not to force optimal performance, even at the risk of creating duplicate document URIs. See "Time vs. Correctness: Using Direct Forest Updates" on page 54. Default: `false`.
-hadoop_conf_dir *string*	When using distributed mode, the Hadoop config directory. For details, see "Configuring Distributed Mode" on page 34.
-archive_metadata_optional *boolean*	When importing documents from a database archive, whether or not to ignore missing metadata files. If this is `false` and the archive contains no metadata, an error occurs. Default: `false`.
-input_compressed *boolean*	Whether or not the source data is compressed. Default: false.
-input_compression_codec *string*	When `-input_compressed` is true, the code used for compression. Accepted values: `zip`, `gzip`.
-input_file_path *string*	A regular expression describing the filesystem location(s) to use for input. For details, see "Regular Expression Syntax" on page 76.
-input_file_pattern *string*	Load only input files that match this regular expression from the path(s) matched by `-input_file_path`. For details, see "Regular Expression Syntax" on page 76. Default: Load all files.
-input_file_type *type*	The input file type. Accepted value: `aggregates`, `archive`, `delimited_text`, `documents`, `sequencefile`. Default: `documents`.
-sequencefile_key_class *string*	When importing Hadoop sequence files, the name of the Java class to use as the input key. Required when using sequence files.

Option	Description
-sequencefile_value_class *string*	When importing Hadoop sequence files, the name of the Java class to use as the input value. Required when using sequence files.
-sequencefile_value_type *string*	When importing Hadoop sequence files, the type of the value data returned by the class named by `-sequencefile_value_class`. Accepted values: `Text`, `BytesWritable`. (Values are case-insensitive). Default: `Text`.
-max_split_size *number*	When importing from files, the maximum number of bytes in one input split. Default: The maximum Long value (`Long.MAX_VALUE`).
-min_split_size *number*	When importing from files, the minimum number of bytes in one input split. Default: 0.
-mode *string*	Ingestion mode. Accepted values: `distributed`, `local`. Distributed mode requires Hadoop. Default: `local`.
-namespace *string*	The default namespace for all XML documents created during loading.
-options_file *string*	Specify an options file pathname from which to read additional command line options. If you use an options file, this option must appear first. For details, see “Options File Syntax” on page 77.
-output_cleandir *boolean*	Whether or not to delete all content in the output database directory prior to loading. Default: `false`.
-output_collections *comma-list*	A comma separated list of collection URIs. Loaded documents are added to these collections.
-output_directory *string*	The destination database directory in which to create the loaded documents. Using this option enables `-fastload` by default, which can cause duplicate URIs to be created. See “Time vs. Correctness: Using Direct Forest Updates” on page 54.

Option	Description
-filename_as_collection *boolean*	Add each loaded document to a collection corresponding to the name of the input file. Useful when splitting an input file into multiple documents. If the filename contains characters not permitted in a URI, those characters are URI encoded. Default: `false`.
-output_language *string*	The `xml:lang` to associate with loaded documents.
-output_permissions *comma-list*	A comma separated list of (`role,capability`) pairs to apply to loaded documents. Default: The default permissions associated with the user inserting the document. Example: `-output_permissions role1,read,role2,update`
-output_quality *string*	The quality of loaded documents. Default: 0.
-output_uri_prefix *string*	URI prefix to the id specified by `-output_idname`. Used to construct output document URIs.
-output_uri_replace *comma-list*	A comma separated list of (`regex,string`) pairs that define string replacements to apply to the URIs of documents added to the database. The replacement strings must be enclosed in single quotes. For example, `-output_uri_replace "regex1,'string1',regext2,'string2'"`
-output_uri_suffix *string*	URI suffix to the id specified by `-output_idname`. Used to construct output document URIs.
-streaming *boolean*	Whether or not to stream documents to MarkLogic Server. Applies only when `-input_file_type` is `documents`.

Option	Description
-thread_count *number*	The number of threads to spawn for concurrent loading. Only available in `local` mode. Default: 4.
-tolerate_errors *boolean*	Applicable only when `-batch_size` is greater than 1. When this option is true and batch size is greater than 1, if an error occurs for one or more documents during loading, only the erroneous documents are skipped; all other documents are inserted into the database. When this option is false or batch size is > 1, errors during insertion can cause all the inserts in the current batch to be rolled back. Default: false.
-transaction_size *number*	The number of requests to MarkLogic Server per transaction. Default: 10.
-xml_repair_level *string*	The degree of repair to attempt on XML documents in order to create well-formed XML. Accepted values: `default`, `full`, `none`. Default: `default`, which depends on the configured MarkLogic Server default XQuery version: In XQuery 1.0 and 1.0-ml the default is none. In XQuery 0.9-ml the default is full.

7.12.4 Export Options

This section summarizes the command line options available with the mlcp `export` command. For examples and use cases, see “Exporting Content from MarkLogic Server” on page 58.

The following command line options are required when using the `export` command:

Option	Description
-host *string*	Hostname of the source MarkLogic Server.
-port *number*	Port number of the source MarkLogic Server. There should be an XDBC App Server on this port, listening on behalf of the target database. The App Server must not be SSL-enabled.
-username *string*	MarkLogic Server user with which to export documents.
-password *string*	Password for the MarkLogic Server user specified with `-username`.

The following table lists optional command line options:

Option	Description
-collection_filter *comma-list*	A comma-separated list of collection URIs. mlcp exports only documents in these collections, plus related metadata. This option may not be combined with `-directory_filter`. Default: All documents and related metadata.
-compress *boolean*	Whether or not to compress the output document. Only applicable when `-output_type` is `document`. Default: `false`.
-conf *filename*	Pass extra setting to Hadoop when using distributed mode. For details, see “Setting Custom Hadoop Options and Properties” on page 35. This option must appear before mlcp-specific options.
-copy_collections *boolean*	When exporting documents to an archive, whether or not to copy collections to the destination. Default: `true`.
-copy_permissions *boolean*	When exporting documents to an archive, whether or not to copy document permissions to the destination. Default: `true`.

Option	Description
-copy_properties *boolean*	When exporting documents to an archive, whether or not to copy properties to the destination. Default: `true`.
-copy_quality *boolean*	When exporting documents to an archive, whether or not to copy document quality to the destination. Default: `true`.
-D *property=value*	Pass a configuration property setting to Hadoop when using distributed mode. For details, see "Setting Custom Hadoop Options and Properties" on page 35. This option must appear before mlcp-specific options.
-directory_filter *comma-list*	A comma-separated list of database directory names. mlcp exports only documents from these directories, plus related metadata. Directory names should usually end with "/". This option may not be combined with `-collection_filter`. Default: All documents and related metadata.
-hadoop_conf_dir *string*	When using distributed mode, the Hadoop config directory. For details, see "Configuring Distributed Mode" on page 34.
-indented *boolean*	Whether to pretty-print XML output. Default: `false`.
-max_split_size *number*	The maximum number of document fragments processed per split. Only applicable in distributed mode. Default: 50000.
-mode *string*	Export mode. Accepted values: `distributed`, `local`. Distributed mode requires Hadoop. Default: `local`.

Option	Description
-options_file *string*	Specify an options file pathname from which to read additional command line options. If you use an options file, this option must appear first. For details, see "Options File Syntax" on page 77.
-output_file_path *string*	Destination directory where the archive or documents are saved. The directory must not already exist.
-output_type *string*	The type of output to produce. Accepted values: `document`, `archive`. Default: `document`.
-thread_count *number*	The number of threads to spawn for concurrent exporting. Only available in `local` mode. Default: 4.

7.12.5 Copy Options

This section summarizes the command line options available with the mlcp `copy` command. For examples and use cases, see "Copying Content Between Databases" on page 61.

The following command line options are required when using the `copy` command:

Option	Description
-input_host *string*	Hostname of the source MarkLogic Server.
-input_port *number*	Port number of the source MarkLogic Server.There should be an XDBC App Server on this port, listening on behalf of the source database. The App Server must not be SSL-enabled.
-input_username *string*	MarkLogic Server user with which to export documents.
-input_password *string*	Password for the MarkLogic Server user specified with `-input_username`.

Option	Description
-output_host *string*	Hostname of the destination MarkLogic Server.
-output_port *number*	Port number of the destination MarkLogic Server.There should be an XDBC App Server on this port, listening on behalf of the destination database. The App Server must not be SSL-enabled.
-output_username *string*	MarkLogic Server user with which to import documents to the destination.
-output_password *string*	Password for the MarkLogic Server user specified with `-output_username`.

The following table lists optional command line options:

Option	Description
-batch_size *number*	The number of documents to load per request to MarkLogic Server. Default: 100.
-collection_filter *comma-list*	A comma-separated list of collection URIs. mlcp exports only documents in these collections, plus related metadata. Directory names should usually end with "/". This option may not be combined with `-directory_filter`. Default: All documents and related metadata.
-conf *filename*	Pass extra setting to Hadoop when using distributed mode. For details, see "Setting Custom Hadoop Options and Properties" on page 35. This option must appear before mlcp-specific options.
-copy_collections *boolean*	Whether to copy document collections from the source database to the destination database. Default: `true`.
-copy_permissions *boolean*	Whether to copy document permissions from the source database to the destination database. Default: `true`.

Option	Description
-copy_properties *boolean*	Whether to copy document properties from the source database to the destination database. Default: `true`.
-copy_quality *boolean*	Whether to copy document quality from the source database to the destination database. Default: `true`.
-D *property=value*	Pass a configuration property setting to Hadoop when using distributed mode. For details, see "Setting Custom Hadoop Options and Properties" on page 35. This option must appear before mlcp-specific options.
-directory_filter *comma-list*	A comma-separated list of database directories. mlcp exports only documents from these directories, plus related metadata. This option may not be combined with `-collection_filter`. Default: All documents and related metadata.
-fastload *boolean*	Whether or not to force optimal performance, even at the risk of creating duplicate document URIs. See "Time vs. Correctness: Using Direct Forest Updates" on page 54. Default: `false`.
-hadoop_conf_dir *string*	When using distributed mode, the Hadoop config directory. For details, see "Configuring Distributed Mode" on page 34.
-max_split_size *number*	The maximum number of document fragments processed per split. Only applicable in distributed mode. Default: 50000.
-mode *string*	Copy mode. Accepted values: `distributed`, `local`. Distributed mode requires Hadoop. Default: `local`.
-options_file *string*	Specify an options file pathname from which to read additional command line options. If you use an options file, this option must appear first. For details, see "Options File Syntax" on page 77.
-output_collections *comma-list*	A comma separated list of collection URIs. Output documents are added to these collections.

Option	Description
-output_permissions *comma-list*	A comma separated list of (`role,capability`) pairs to apply to loaded documents. Default: The default permissions associated with the user inserting the document. Example: `-output_permissions role1,read,role2,update`
-output_quality *string*	The quality to assign to output documents.
-output_uri_prefix *string*	URI prefix to the id specified by `-output_idname`. Used to construct output document URIs.
-output_uri_replace *comma-list*	A comma separated list of (`regex,string`) pairs that define string replacements to apply to the URIs of documents added to the database. The replacement strings must be enclosed in single quotes. For example, `-output_uri_replace "regex1,'string1',regext2,'string2'"`
-output_uri_suffix *string*	URI suffix to the id specified by `-output_idname`. Used to construct output document URIs.
-thread_count *number*	The number of threads to spawn for concurrent copying. Only available in `local` mode. Default: 4.
-transaction_size *number*	When loading documents into the destination database, the number of requests to MarkLogic Server in one transaction. Default: 10.

7.12.6 Regular Expression Syntax

For `-input_file_path`, use the regular expression syntax outlined here:

> http://hadoop.apache.org/docs/stable/api/org/apache/hadoop/fs/FileSystem.html#globStatus(org.apache.hadoop.fs.Path)

For all other options that use regular expressions, such as `-input_file_pattern`, use the Java regular expression language. Java's pattern language is similar to the Perl pattern language. For details on the grammar, see the documentation for the Java class `java.util.regex.Pattern`:

> http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html

For a tutorial on the expression language, see http://docs.oracle.com/javase/tutorial/essential/regex/.

7.12.7 Options File Syntax

You can specify mlcp options using an options file, in addition to using command line options by using `-options_file`. If you use an options file, it must be the first option on the command line. The mlcp command (import, export, copy) can also go inside the options file. For example:

```
$ mlcp.sh -options_file my_options.txt -input_file_path /example
```

An options file has the following contents:

- Each line contains either a command name, an option, or an option value, ordered as they would appear on the command line.
- Comments begin with "#" and must be on a line by themselves.
- Blank lines, leading whitespace, and trailing whitespace are ignored.

For example, if you frequently use the same MarkLogic Server connection information (host, port, username, and password), you can put the this information into an options file:

```
$ cat my-conn.txt
# my connection info
-host
localhost
-port
8006
-username
me
-password
my_password

# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -options_file my-conn.txt \
    -input_file_path /space/examples/all.zip
```

This is equivalent to the following command line without an options file:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8006 -username me \
    -password my_password -input_file_path /space/examples/all.zip
```

You can also include a command name (import, export, or copy) as the first non-comment line in an options file:

```
# my connection info for import
import
-host
localhost
-port
8006
-username
me
```

```
-password
my_password
```

8.0 Loading Content Using WebDAV

If you have configured a WebDAV server, you can use a WebDAV client to load documents into the database. WebDAV clients such as Windows Explorer allow drag and drop access to documents, just like any other documents on the filesystem. There are a number of WebDAV clients available for various platforms and information can be found by searching the Internet for WebDAV clients. Some MarkLogic users have had good success with BitKinex and with NetDrive from Novell.

For details on setting up MarkLogic WebDAV servers, see WebDAV Servers in the *Administrator's Guide*.

Directories are required for WebDAV clients to see documents. See Directories and WebDAV Servers in the *Application Developer's Guide*.

For an example of using a WebDAV client with the Default Conversion Option (a CPF example application), see Simple Drag-and-Drop Conversion in the *Content Processing Framework Guide*.

9.0 Repairing XML Content During Loading

MarkLogic Server can perform the following types of XML content repair during content loading:

- Correct content that does not conform to the well-formedness rules in the XML specification
- Modify inconsistently structured content according to a specific XML schema
- Assign namespaces and correct unresolved namespace bindings
- Restructure content using XPath or XQuery

Not all programming language interfaces support the full spectrum of XML content repair. MarkLogic Server does not validate content against predetermined XSchema (DDML) or DTDs.

This chapter includes the following topics:

- Programming Interfaces and Supported Content Repair Capabilities
- Enabling Content Repair
- Auto-Close Repair of Empty Tags
- Schema-Driven Tag Repair
- Load-Time Default Namespace Assignment
- Load-Time Namespace Prefix Binding
- Query-Driven Content Repair

9.1 Programming Interfaces and Supported Content Repair Capabilities

The MarkLogic programming interfaces support repair options as described in the following table:

Programming Interface	Content Repair Capabilities	More Details
MarkLogic Connector for Hadoop	Set the repair level.	*MarkLogic Connector for Hadoop Developer's Guide*
MarkLogic Content Pump	Tag repair and schema-driven repair, namespace prefix binding.	`-xml_repair_level` option. See "Import Options" on page 64
Information Studio	Tag repair, schema-driven repair, namespace prefix binding, default namespace.	Ingestion configuration option. See Configuring Ingestion Options in the *Information Studio Developer's Guide*
MarkLogic Java API	Tag repair, schema-driven repair, namespace prefix binding.	*Java Application Developer's Guide*
MarkLogic REST API	General-purpose tag repair and schema-driven repair, namespace prefix binding.	`repair` parameter on `PUT /documents` in the MarkLogic REST API
XCC	General-purpose tag repair and schema-driven repair, namespace prefix binding.	`DocumentRepairLevel` enumeration class
XQuery	All types described in this chapter.	`<repair>` parameter in the options node of `xdmp:document-load`. Also see the *MarkLogic XQuery and XSLT Function Reference*

9.2 Enabling Content Repair

The tag repair, schema-driven repair, and namespace prefix binding mechanisms are enabled using an option to the various content loading functions as listed above.

When no repair option is explicitly specified, the default is implicitly specified by the XQuery version of the caller. In XQuery 1.0 and 1.0-ml the default is `none`. In XQuery 0.9-ml the default is `full`.

Tag repair, schema-driven repair, and namespace prefix binding can be performed on all XML documents loaded from external sources. This includes documents loaded using the XQuery built-in functions, XCC document insertion methods, or the Java or REST client APIs.

9.3 General-Purpose Tag Repair

MarkLogic Server can apply a general-purpose, stack-driven tag repair algorithm to every XML document loaded from an external source. The algorithm is triggered by encountering a closing tag (for example, `</tag>`) that does not match the most recent opening tag on the stack.

9.3.1 How General-Purpose Tag Repair Works

Consider the following simple document markup example:

```
<p>This is <b>bold and <i>italic</i></b> within the paragraph.</p>
```

Each variation below introduces a tagging error common to hand-coded markup:

```
<p>This is <b>bold and <i>italic</b> within the paragraph.</p>

<p>This is <b>bold and <i>italic</i></b></u> within the paragraph.</p>
```

In the first variation, the `italic` element is never closed. And in the second, the `underline` element is never opened.

When MarkLogic Server encounters an unexpected closing tag, it performs one of the following actions:

- **Rule 1**: If the QName (both the tag's namespace *and* its localname) of the unexpected closing tag matches the QName of a tag opened earlier and not yet closed, the loader automatically closes all tags until the matching opening tag is closed.

  Consequently, in the first sample tagging error shown above, the loader automatically closes the italic element when it encounters the tag closing the bold element:

  ```
  <p>This is <b>bold and <i>italic</i></b> within the paragraph.</p>
  ```

  The bold characters in the markup shown above indicate the close tag dynamically inserted by the loader.

- **Rule 2**: If there is no match between the QName of the unexpected closing tag and all previously opened tags, the loader ignores the closing tag and proceeds.

  Consequently, in the second tagging error shown above, the loader ignores the "extra" underline closing tag and proceeds as if it is not present:

  ```
  <p>This is <b>bold and <i>italic</i></b></u> within the paragraph.</p>
  ```

  The italic tag shown above indicates the closing tag that the loader is ignoring.

Both rules work together to repair even more complex situations. Consider the following variation, in which the bold and italic closing tags are mis-ordered:

```
<p>This is <b>bold and <i>italic</b></i> within the paragraph.</p>
```

In this circumstance, the first rule automatically closes the italic element when the closing bold tag is encountered. When the closing italic tag is encountered, it is simply discarded as there are no previously opened italic tags still on the loader's stack. The result is more than likely what the markup author intended:

```
<p>This is <b>bold and <i>italic</i></b> within the paragraph.</p>
```

9.3.2 Pitfalls of General-Purpose Tag Repair

While these two general repair rules produce sound results in most situations, their application can lead to repairs that may not match the original intent. Consider the following examples.

1. This snippet contains a markup error – the bold element is never closed.

   ```
   <p>This is a <b>bold and <i>italic</i> part of the paragraph.</p>
   ```

   The general-purpose repair algorithm fixes this problem by inserting a closing bold tag just in front of the closing paragraph tag, because this is the point at which it becomes apparent that there is a markup problem:

   ```
   <p>This is a <b>bold and <i>italic</i> part of the paragraph.</b></p>
   ```

   In this situation, the entire remainder of the paragraph is emboldened, because it is not otherwise apparent where the tag should have been closed. In fact, for cases other than this simple example, even a human is not always able to make the right decision.

2. Rule 1 can also cause significant "unwinding" of the stack if a tag, opened much earlier in the document, is mistakenly closed mid-document. Consider the following markup error where `</d>` is mistyped as `</b>`.

   ```
   <a>
     <b>
       <c>
         <d>...content intended for d...</b>
         ...content intended for c...
       </c>
       ...content intended for b...
     </b>
     ...content intended for a...
   </a>
   ```

   The erroneous `</b>` tag triggers rule 1 and the system closes all intervening tags between `<b>` and `<d>`. Rule 2 then discards the actual close tags for `<b>` and `<c>` that have now been made redundant (since they have been closed by rule 1). This results in an incorrectly

"flattened" document as shown below (some indentation and line breaks have been added for illustrative purposes):

```
<a>
  <b>
    <c>
      <d>...content intended for d...</d>
    </c>
  </b>
  ...content intended for c...
  ...content intended for b...
  ...content intended for a...
</a>
```

General-purpose tag repair is not always able to correctly repair structure problems, as shown in the preceding examples. MarkLogic offers additional content repair capabilities that can be used to repair a wider range of problems, including the examples above. These advanced content repair techniques are described in the following sections.

9.3.3 Limitations

This section describes some known limitations of general-purpose tag repair.

9.3.3.1 XQuery Functions

For functions where the XML node provided as a parameter is either dynamically generated by the query itself (and is consequently guaranteed to be well-formed) or is explicitly defined within the XQuery code (in which case the query is not successfully parsed for execution unless it is well-formed), general-purpose tag repair is not performed. This includes XML content loaded using the following functions:

- `xdmp:document-insert`
- `xdmp:node-replace`
- `xdmp:node-insert-before`
- `xdmp:node-insert-after`
- `xdmp:node-insert-child`

9.3.3.2 Root Element

General-purpose tag repair does not insert a missing closing root element tag into an XML document.

9.3.3.3 Previous Marklogic Versions

Versions of MarkLogic Server 2.0 and earlier would repair missing root elements, making it effectively impossible to identify truncated source content. Later versions of MarkLogic Server reports an error in these conditions.

9.3.4 Controlling General-Purpose Tag Repair

MarkLogic Server enables you to enable or disable general-purpose tag repair during any individual document load using an optional repair parameter. The specific parameter is language specific. For example, if you use XQuery `xdmp:document-load`, `xdmp:unquote` functions, you can use the `repair` parameter on the options node and specify a value of `full` or `none`. See the language specific documentation for more details.

9.4 Auto-Close Repair of Empty Tags

Empty tag auto-close is a special case of schema-driven tag repair and is supported in all versions of MarkLogic Server. This repair mechanism automatically closes tags that are identified as empty tags in a specially-constructed XML schema.

This approach addresses a common problem found in SGML and HTML documents. SGML and HTML both regard tags as markup rather than as the hierarchical element containers defined by the XML specification. In both the SGML and HTML worlds, it is acceptable to use a tag as an indication of some formatting directive, without any need to close the tag. This frequently results in the liberal use of empty tags within SGML and HTML content.

For example, an `<hr>` tag in an HTML document indicates a horizontal rule. Because there is no sense to containing anything within a horizontal rule, the tag is interpreted by browsers as an empty tag. Consequently, while HTML documents may be littered with `<hr>` tags, you rarely find a `</hr>` tag or even a `<hr/>` tag unless someone has converted the HTML document to be XHTML-compliant. The same can occur with `<img>` and `<meta>` tags, to name just two. In SGML documents, you can easily find `<pgbrk>`, `<xref>` and `<graphic>` used similarly.

Applying this type of content repair enables you to avoid the false nesting of content within otherwise unclosed empty tags.

9.4.1 What Empty Tag Auto-Close Repair Does

Consider the following simple SGML document snippet:

```
<book>
<para>This is the first paragraph.</para>
<pgbrk>
<para>This paragraph has a cross-reference <xref id="f563t001"> in some
<italic>italic</italic> text.</para>
</book>
```

This snippet incorporates two tags, `<pgbrk>` and `<xref>`, that are traditionally viewed as empty tags. Working under default settings, MarkLogic Server views each of these two tags as opening tags that at some point later in the document will be closed, and consequently incorrectly views the following content as children of those tags. This results in a falsely nested document (indentation and line breaks added for clarification):

```
<book>
  <para>
```

```
    This is the first paragraph.
  </para>
  <pgbrk>
    <para>
      This paragraph has a cross-reference
      <xref id="f563t001">
        in some
        <italic>italic</italic>
        text.
      </xref>
    </para>
  </pgbrk>
</book>
```

The bold characters in the markup shown above indicate closing tags automatically inserted by the general-purpose tag repair algorithm.

This example demonstrates how unclosed empty tags can distort the structure of a document. Imagine how much worse this example could get if it had fifty `<pgbrk>` tags in it.

To understand the ramifications of this, consider how the markup applied above is processed by a query that specifies an XPath such as `/doc/para`. The first paragraph matches this XPath, but the second does not, because it has been loaded incorrectly as the child of a `pgbrk` element. While alternative XPath expressions such as `/doc//para` gloss over this difference, it is better to load the content correctly in the first place (indentation and line breaks added for clarification):

```
<book>
  <para>
    This is the first paragraph.
  </para>
  <pgbrk/>
  <para>
    This paragraph has a cross-reference
    <xref id="f563t001"/>
    in some
    <italic>italic</italic>
    text.
  </para>
</book>
```

9.4.2 Defining a Schema to Support Empty Tag Auto-Close Repair

To use empty tag auto-close repair, you first define an XML schema that specifies which tags should be assumed to be empty tags. Using this information, when MarkLogic Server is loading content from an external source, it automatically closes these tags as soon as they are encountered. If some of the specified tags are, in fact, accompanied by closing tags, these closing tags are discarded by the general-purpose tag repair algorithm.

Here is an example of a schema that instructs the loader to treat as empty tags any `<xref>`, `<graphic>` and `<pgbrk>` tags found in documents governed by the `http://www.mydomain.com/sgml` namespace:

```
<xs:schema
  targetNamespace="http://www.mydomain.com/sgml"
  xsi:schemaLocation="http://www.w3.org/2001/XMLSchema XMLSchema.xsd"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xs:complexType name="empty"/>
  <xs:element name="xref" type="empty"/>
  <xs:element name="graphic" type="empty"/>
  <xs:element name="pgbrk" type="empty"/>
</xs:schema>
```

If the sample SGML document shown earlier is loaded under the control of this schema, it is repaired correctly.

To use XML schemas for content repair, two things are required:

- The schema must be loaded into MarkLogic Server.
- The content to be loaded must properly reference the schema at load-time.

9.4.3 Invoking Empty Tag Auto-Close Repair

There are multiple ways to invoke the empty tag auto-close functionality. The recommended procedure is the following:

1. Write an XML schema that specifies which tags should be treated as empty tags. The schema shown in the preceding section, “Defining a Schema to Support Empty Tag Auto-Close Repair” on page 86, is a good starting point.

2. Load the schema into MarkLogic. See Loading Schemas in the *Application Developer’s Guide* for instructions.

3. Make sure that the content to be loaded references the namespace of the applicable schema that you have loaded into MarkLogic. For the schema shown above, the document’s root element could take one of two forms.

   In the first form, the document implicitly references the schema through its namespace:

   ```
   <document
     xmlns="http://www.mydomain.com/sgml">
   ```

```
  ...
</document>
```

MarkLogic Server automatically looks for a matching schema whenever a document is loaded.

In the second form, one of multiple matching schemas can be explicitly referenced by the document being loaded:

```
<document
  xmlns="http://www.mydomain.com/sgml"
  xsi:schemaLocation="http://www.mydomain.com/sgml /sch/SGMLEmpty.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema">
  ...
</document>
```

This example explicitly references the schema stored at URI `/sch/SGMLEmpty.xsd` in the current schema database. If there is no schema stored at that URI, or the schema stored at that URI has a target namespace other than `http://www.mydomain.com/sgml`, no schema is used.

See Loading Schemas in the *Application Developer's Guide* for an in-depth discussion of the precedence rules that are applied in the event that multiple matching schemas are found.

4. Load the content using `xdmp:document-load` or one of the other language interface document insertion methods.

After the content is loaded, you can inspect it to see that the content repair was performed. If empty tag auto-close repair was not applied, then you should troubleshoot the location, naming and cross-referencing of your schema, as this is the most likely source of the problem.

When it is not feasible to modify your content so that it properly references a namespace in its root element, there are other approaches that can yield the same result:

1. Write an XMLschema that specifies which tags should be treated as empty tags. Because the root `xs:schema` element lacks a `targetNamespace` attribute, the document below specifies a schema that applies to documents loaded in the unnamed namespace:

```
<xs:schema
  xsi:schemaLocation="http://www.w3.org/2001/XMLSchema XMLSchema.xsd"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xs:complexType name="empty"/>
  <xs:element name="xref" type="empty"/>
  <xs:element name="graphic" type="empty"/>
  <xs:element name="pgbrk" type="empty"/>
</xs:schema>
```

2. Load the schema into MarkLogic, remembering the URI name under which you loaded the schema. See Loading Schemas in the *Application Developer's Guide* for instructions on properly loading schema in MarkLogic Server.

3. Construct an XQuery statement that temporarily imports the schema into the appropriate namespace and loads the content within that context.

   - A simple example of importing a schema into the unnamed namespace might look like the following:

     ```
     xquery version "0.9-ml"
     import schema namespace "myNS" at "schema-uri-you-specified-in-step-2";
     xdmp:document-load("content-to-be-repaired.sgml", ...)
     ```

     Be careful to restrict the content loading operations you carry out within the context of this `import schema` directive, as all documents loaded in the unnamed namespace are filtered through the "empty tag auto close" repair algorithm under the control of this schema.

     Note: The target namespace specified in the `import schema` prolog statement and in the schema document itself must be the same, otherwise the schema import fails silently.

4. Run the query shown above to load and repair the content.

9.4.4 Scope of Application

Once a schema is configured and loaded for empty tag auto-closing, any content that references that schema and is loaded from an external source is automatically repaired as directed by that schema.

9.4.5 Disabling Empty Tag Auto-Close

There are several ways to disable load-time empty tag auto-close repair:

1. Disable content repair at load-time using the applicable option for your chosen language interface.

2. Remove the corresponding schema from the database and ensure that none of the content to be loaded in the future still references that schema.

3. Modify the referenced schema to remove the empty tag definitions.

Removing the schema from the database does not impact documents already loaded under the rubric of that schema, at least with respect to their empty tags being properly closed. To the extent that the schema in question contains other information about the content that is used during query processing, you should consider the removal of the schema from the database carefully.

9.5 Schema-Driven Tag Repair

MarkLogic Server supports the use of XML schemas for more complex schema-driven tag repair. This enables you to use XML schemas to define a set of general rules that govern how various elements interact hierarchically within an XML document.

9.5.1 What Schema-Driven Tag Repair Does

For example, consider the following SGML document snippet:

```
<book>
<section><para>This is a paragraph in section 1.
<section><para>This is a paragraph in section 2.
</book>
```

This snippet illustrates one of the key challenges created by interpreting markup languages as XML. Under default settings, the server repairs and loads this content as follows (indentation and line breaks added for clarification):

```
<book>
  <section>
    <para>
    This is a paragraph in section 1.
      <section>
        <para>This is a paragraph in section 2.</para>
      </section>
    </para>
  </section>
</book>
```

The repaired content shown above is almost certainly not what the author intended. However, it is all that the server can accomplish using only general-purpose tag repair.

Schema-driven content repair improves the situation by allowing you to indicate constraints in the relationships between elements by using an XML schema. In this case, you can indicate that a `<section>` element may only contain `<para>` elements. Therefore, a `<section>` element cannot be a child of another `<section>` element. In addition, you can indicate that `<para>` element is a simple type that only contains text. Using the schema, MarkLogic Server can improve the quality of content repair that it performs. For example, the server can use the schema to know that it should check to see if there is an open `<section>` element on the stack whenever it encounters a new `<section>` element.

The resulting repair of the SGML document snippet shown above is closer to the original intent of the document author:

```
<book>
  <section>
    <para>
      This is a paragraph in section 1.
    </para>
  </section>
  <section>
    <para>
      This is a paragraph in section 2.
    </para>
  </section>
</book>How it works
```

To take advantage of schema-driven tag repair, you must first define an XML schema that describes the constraints on the relationships between elements. Using this information, when tMarkLogic Server loads content from an external source, it automatically closes tags still open on its stack when it encounters an open tag that would violate the specified constraints.

Unlike general-purpose tag repair, which is triggered by unexpected *closing* tags, schema-driven tag repair is triggered by unexpected *opening* tags, so the two different repair models interoperate cleanly. In the worst case, schema-driven tag repair may, as directed by the governing schema for the document being loaded, automatically close an element sooner than that element is explicitly closed in the document itself. This case only occurs when the relationship between elements in the document is at odds with the constraints described in the schema, in which case the schema is used as the dominating decision factor.

The following is an example of a schema that specifies the following constraints:

- `<book>` elements in the `http://www.mydomain.com/sgml` namespace can only contain `<section>` elements.
- `<section>` elements in the `http://www.mydomain.com/sgml` namespace can only contain `<para>` elements.
- `<para>` elements in the `http://www.mydomain.com/sgml` namespace can only contain text.

```
<xs:schema
  targetNamespace="http://www.mydomain.com/sgml"
  xsi:schemaLocation="http://www.w3.org/2001/XMLSchema XMLSchema.xsd"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <xs:complexType name="book">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="section"/>
    </xs:choice>
  </xs:complexType>

  <xs:complexType name="section">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="para"/>
    </xs:choice>
  </xs:complexType>

  <xs:element name="book" type="book"/>
  <xs:element name="section" type="section"/>
  <xs:element name="para" type="xs:string"/>
</xs:schema>
```

If the sample SGML document shown above is loaded under the control of this simple schema, it is corrected as specified.

To make this happen, two things are required:

- The schema must be loaded into MarkLogic Server.
- The content to be loaded must properly reference this schema at load-time.

9.5.2 How to Invoke Schema-Driven Tag Repair

There are multiple ways to do schema-driven correction. The recommended procedure is the following:

1. Write an XML schema that describes the relationships between the elements.

2. Load the schema into MarkLogic Server. See Loading Schemas in the *Application Builder Developer's Guide* for instructions.

3. In the content that you need to load, ensure that the root element properly references the appropriate schema. See "Invoking Empty Tag Auto-Close Repair" on page 87 for examples of referencing the XML schema from inside the content.

4. Load the content using `xdmp:document-load` or any of the other available document insertion methods.

After the content is loaded, you can inspect it to see that the content repair was performed. If the appropriate content repair did not occur, then you should troubleshoot the placement, naming and cross-referencing of your schema.

If it is not feasible to modify the content so that it properly references the XML schema in its root element, there are other approaches that can yield the same result:

1. Write a schema that describes the relationships between the elements, and omit a `targetNamespace` attribute from its `xs:schema` root element.

2. Load the schema into MarkLogic Server, remembering the URI name under which you store the schema. See Loading Schemas for instructions on properly loading schema in MarkLogic Server.

3. Construct an XQuery statement that temporarily imports the schema into the appropriate namespace and loads the content within that context. Following is a simple example of importing a schema into the unnamed namespace:

   ```
   xquery version "0.9-ml"
   import schema namespace "myNS" at "schema-uri-you-specified-in-step-1";
   xdmp:document-load("content-to-be-repaired.sgml", ...)
   ```

   Be careful to restrict the content loading operations you carry out within the context of this `import schema` directive, as all documents loaded are filtered through the same schema-driven content repair algorithm.

   Note: The target namespace specified in the `import schema` prolog statement and in the schema document itself must be the same, otherwise the schema import fails silently.

4. Run the query shown above to load and repair the content.

9.5.3 Scope of Application

Once a schema has been configured and loaded for schema-driven tag repair, any content that references that schema and is loaded from an external source is automatically repaired as directed by that schema.

9.5.4 Disabling Schema-Driven Tag Repair

There are several ways to turn off load-time schema-driven tag repair:

1. Disable content repair at load-time using the appropriate parameter for your chosen content loading mechanism.

2. Remove the corresponding schema from the database and ensure that none of the content loaded in the future references that schema.

3. Modify the referenced schema to remove the empty tag definitions.

Removing the schema from the database does not impact documents already loaded under the rubric of that schema. To the extent that the schema in question contains other information about the content that is used during query processing, you should consider the removal of the schema from the database carefully.

9.6 Load-Time Default Namespace Assignment

When documents are loaded into MarkLogic, every element is stored with a QName comprised of a namespace URI and a localname.

However, many XML files are authored without specifying a default namespace or a namespace for any of their elements. When these files are loaded from external sources , MarkLogic applies the default unnamed namespace to all the nodes that do not have an associated namespace.

In some situations this is not the desired result. Once the document is loaded without a specified namespace, it is difficult to remap each QName to a different namespace. It is better to load the document into MarkLogic Server with the correct default namespace in the first place.

The best way to specify a default namespace for a document is to add a default namespace attribute to the document's root node directly. When that is not possible, MarkLogic's load-time namespace substitution capability offers a good solution. If you are using XQuery or XCC for your document loading, you can specify a default namespace for the document at load-time, provided that the document root node does not already contain a default namespace specification.

Note: This function is performed as described below if a default namespace is specified at load time, even if content repair is turned off.

Note: The REST and Java client APIs do not provide a default namespace option. When you use these APIs for your document loading, it is best to add the appropriate namespace attribute to your documents before loading them to the database.

9.6.1 How Default Namespace Assignments Work

The `xdmp:document-load` function and the XCC `setNamespace` method (in the `ContentCreateOptions` class) allow you to optionally specify a namespace as the default namespace for an individual document loading operation.

MarkLogic uses that namespace definition as follows:

Rule 1: If the root node of the document does not contain the default namespace attribute, the server uses the provided namespace as the default namespace for the root node. The appropriate namespaces of descendant nodes are then determined through the standard namespace rules.

Rule 2: If the root node of the document incorporates a default namespace attribute, the server ignores the provided namespace.

Note that rule 2 means that the default namespace provided at load time cannot be used to override an explicitly specified default namespace at the root element

9.6.2 Scope of Application

You can specify default namespaces at load-time when you use XQuery, XCC, or Information Studio to load content. See the corresponding documentation for further details.

9.7 Load-Time Namespace Prefix Binding

The original XML specifications allow the use of colons in element names, for example, `<myprefix:a>`. However, according to the XML Namespace specifications (developed after the initial XML specifications), the string before a colon in an element name is interpreted as a namespace prefix. The use of prefixes that are not bound to namespaces is deemed as non-compliant with the XML Namespace specifications.

Prior to version 2.1, MarkLogic Server dropped unresolved prefixes from documents loaded into the database in order to conform to the XML Namespace specifications. Consider a document named `mybook.xml` that contains the following content:

```
<publisher:book>
  <section>
    This is a section.
  </section>
</publisher:book>
```

If `publisher` is not bound to any namespace, `mybook.xml` is loaded into the database as:

```
<book>
  <section>
    This is a section.
  </section>
<book>
```

Starting in 2.1, MarkLogic Server supports more powerful correction of XML documents with unresolved namespace bindings. If content repair is on, `mybook.xml` is loaded with a namespace binding added for the publisher prefix.

```
<publisher:book
 xmlns:publisher="appropriate namespace-see details below">
  <section>
    This is a section.
  </section>
</publisher:book>
```

If content repair is off, MarkLogic Server returns an error if unresolved namespace prefixes are encountered at load time.

9.7.1 How Load-Time Namespace Prefix Binding Works

If content repair is enabled, MarkLogic can create namespace bindings at load time for namespace prefixes that would otherwise be unresolved.

Namespace prefixes are resolved using the rules below. The rules are listed in order of precedence:

Rule 1: When the prefix is specified in the document, that binding is retained. In the following example, the binding for `publisher` to `"http://publisherA.com"` is specified in the document and is retained.

```
<publisher:book xmlns:publisher="http://publisherA.com">
  <section>
    This is a section.
  </section>
</publisher:book>
```

Rule 2: When the prefix is declared in the XQuery environment, that binding is used. For example, suppose that `mybook.xml`, the document being loaded, contains the following content:

```
<publisher:book>
  <section>
    This is a section.
  </section>
</publisher:book>
```

In addition, suppose that `publisher` is bound to `http://publisherB.com` in the XQuery environment:

```
declare namespace publisher = "http://publisherB.com"

xdmp:document-load("mybook.xml")
```

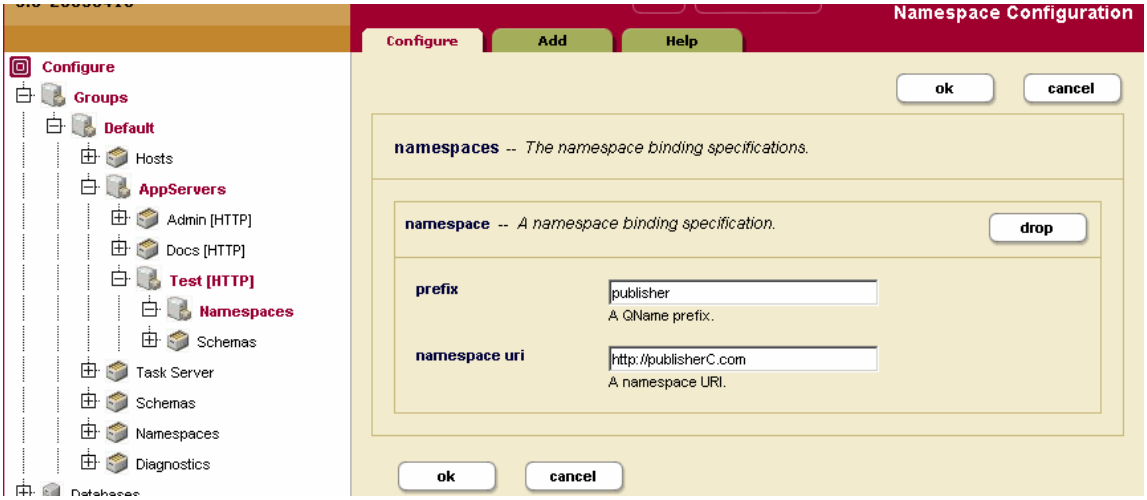
The code snippet loads the `mybook.xml` as:

```
<publisher:book xmlns:publisher="http://publisherB.com">
  <section>
    This is a section.
  </section>
</publisher:book>
```

Note: This rule only applies in the XQuery environment.

Rule 3: If the prefix is declared in the Admin Interface for the HTTP or XDBC server through which the document is loaded, that binding is used.

For example, imagine a scenario in which the namespace prefix `publisher` is defined on the HTTP server named `Test`.



Then, suppose that the following code snippet is executed on `Test`:

```
xdmp:document-load("mybook.xml")
```

The initial document `mybook.xml` as shown in the second case is loaded as:

```
<publisher:book xmlns:publisher="http://publisherC.com">
  <section>
    This is a section.
  </section>
</publisher:book>
```

Rule 4: If no binding for the prefix is found, the server creates a namespace that is the same as the prefix and binds it to the prefix. In this instance, `mybook.xml` is loaded as:

```
<publisher:book xmlns:publisher="publisher">
  <section>
    This is a section.
  </section>
</publisher:book>
```

9.7.2 Interaction with Load-Time Default Namespace Assignment

While both load-time default namespace assignment and load-time namespace prefix binding involve document namespaces, the two features work independently. The former allows the assignment of a default namespace at the root element level, while the latter creates bindings for namespaces that are otherwise unresolved.

Consider the examples below:

1. This document has neither a binding for the `publisher` prefix, nor a default namespace.

```
<publisher:book>
  <section>
    This is a section.
  </section>
</publisher:book>
```

   Suppose a default namespace `http://publisher.com/default-namespace` is specified at load time, and the publisher prefix resolves to `http://publisher.com/prefix` according to the rules described in the previous section. The document is loaded as:

```
<publisher:book xmlns:publisher="http://publisher.com/prefix"
 xmlns="http://publisher.com/default-namespace">
  <section>
    This is a section.
  </section>
</publisher:book>
```

   In this case, `<book>` is in the `"http://publisher.com/prefix"` namespace, while `<section>` is in the `"http://publisher.com/default-namespace"` namespace.

2. This document has a binding for the `publisher` prefix, but does not specify a default namespace in the root node.

```
<publisher:book xmlns:publisher="http://publisher.com/prefix">
  <section>
    This is a section.
  </section>
</publisher:book>
```

   If `http://publisher.com/default-namespace` is specified as the default namespace at load time, the loaded document is the same as the document loaded in the example above.

3. This document specifies a default namespace, but does not contain a binding for the publisher prefix, this time, associated with the `<section>` element.

```
<book xmlns="http://publisher.com/original-namespace">
  <publisher:section>
    This is a section.
    <paragraph>
      This is a paragraph.
    </paragraph>
```

```
  </publisher:section>
</book>
```

If a default namespace `http://publisher.com/default-namespace` is specified at load time, it is ignored. Assume that publisher resolves to `publisher`. The document is loaded as shown below:

```
<book xmlns="http://publisher.com/original-namespace">
  <publisher:section xmlns:publisher="publisher">
    This is a section.
    <paragraph>
      This is a paragraph.
    </paragraph>
  </publisher:section>
</book>
```

In this case, the `<book>` and `<paragraph>` elements are in the default namespace `http://publisher.com/original-namespace`, while the `<section>` element is in the `publisher` namespace.

9.7.3 Scope of Application

If content repair is enabled, MarkLogic attempts to create bindings for unresolved namespace prefixes as a form of content repair for all documents loaded from external sources according to the rules described in “How Load-Time Namespace Prefix Binding Works” on page 96.

9.7.4 Disabling Load-Time Namespace Prefix Binding

MarkLogic enables you to disable content repair during any individual document load using a language specific repair parameter. See “Programming Interfaces and Supported Content Repair Capabilities” on page 81.

9.8 Query-Driven Content Repair

The content repair models described above influence the content as it is loaded, trying to ensure that the structure of the poorly or inconsistently formatted content is as close to the author's intent as possible when it is first stored in the database.

When a situation requires content repair that is beyond the scope of some combination of these four approaches, MarkLogic's schema-independent core makes XQuery itself a powerful content repair mechanism.

Once a document is loaded into MarkLogic, queries can be written to specifically restructure the content as required, without needing to reconfigure the database. Two approaches to query-driven content repair -- point repair and document walkers -- are described in the following sections. If you want to do something similar from other languages, use a transformation, a feature of the REST and Java client API's that lets you install XQuery or XSLT that you can use during document loading and retrieval.

9.8.1 Point Repair

Point repair uses XPath-based queries to identify document subtrees of interest, create repaired content structures from the source content, and then call `xdmp:node-replace` to replace the document subtree of interest. A simple example of such a query is shown below:

```
for $node-to-be-repaired in doc($uri-to-be-repaired)//italic
return
  xdmp:node-replace($node-to-be-repaired,
    <i>{ $node-to-be-repaired/* }</i>)
```

This example code finds every element with localname `italic` in the default element namespace and changes its QName to localname `i` in the default element namespace. All of the element's attributes and descendants are inherited as is.

An important constraint of the XQuery shown above lies in its assumption that italic elements cannot be descendants of other italic elements, a constraint that should be enforced at load-time using schema-driven content repair. If such a situation occurs in the document specified by `$uri-to-be-repaired`, the above XQuery generates an error.

9.8.2 Document Walkers

Document walkers use recursive descent document processing functions written in XQuery to traverse either the entire document or a subtree within it, create a transformed (and appropriately repaired) version of the document, and then call `xdmp:document-insert` or `xdmp:node-replace` to place the repaired content back into the database.

Queries involving document traversal are typically more complex than point repair queries, because they deal with larger overall document context. Because they can also traverse the entire document, the scope of repairs that they can address is also significantly broader.

The `walk-tree` function shown below uses a recursive descent parser to traverse the entire document:

```
xquery version "1.0-ml";
declare function local:walk-tree(
  $node as node())
as node()
{
  if (xdmp:node-kind($node) = "element") then
    (: Reconstruct node and its attributes; descend to its children :)
    element { fn:node-name($node) }  {
      $node/@*,
      for $child-node in $node/node()
      return
        local:walk-tree($child-node)
    }
  else if (xdmp:node-kind($node) = "comment" or
```

```
            xdmp:node-kind($node) = "processing-instruction" or
            xdmp:node-kind($node) = "text") then
      (: Return the node as is :)
      $node
    else if (xdmp:node-kind($node) = "document") then
      document {
      (: Start descent from the document node's children :)
         for $child-node in $node/node()
         return
           local:walk-tree($child-node)
      }
    else
      (: Should never get here :)
      fn:error(
        fn:concat("Error:  Could not process node of type '",
               xdmp:node-kind($node), "'")
      )
};

let $node := text {"hello"}
return
local:walk-tree($node)
(: returns the text node containing the string "hello" :)
```

This function can be used as the starting point for any content repair query that needs to walk the entire document in order to perform its repair. By inserting further checks in each of the various clauses, this function can transform both the structure and the content. For example, consider the following modification of the first `if` clause:

```
if (xdmp:node-kind($node) = "element") then
  (: Reconstruct node and its attributes; descend to its children :)
  element {
    if (fn:local-name($node) != "italic") then
      fn:node-name($node)
    else
      fn:QName(fn:namespace-uri($node), "i")
  } {
    $node/@*,
    for $child-node in $node/node()
    return
      local:walk-tree($child-node)
  }
```

Inserting this code into the `walk-tree` function enables the function to traverse a document, finding any element whose local-name is `italic`, regardless of that element's namespace, and change that element's local-name to `i`, keeping its namespace unchanged.

You can use the above document walker as the basis for complex content transformations, effecting content repair using the database itself as the repair tool once the content has been loaded into the database.

Another common design pattern for recursive descent is to use a `typeswitch` expression. For details, see Transforming XML Structures With a Recursive typeswitch Expression in the *Application Developer's Guide*.

10.0 Modifying Content During Loading

Content can go through many stages before it is ready to use in an application. These stages might include modifying the content so that it is well-formed XML, transforming one XML structure to another, or combining the content with other content or information. The process of content going from one stage to another is called *content processing*.

Content processing can be very simple or extremely complex. You might decide to add a timestamp to a document and define a content processing stage to add the timestamp. You might have a process that translates the text from one language to another. Often, many of these stages combined together form an overall set of content processing work you need to do on a document.

While the range of problems that can be addressed is virtually unlimited, there are several core content processing capabilities required to address many of the wide-ranging issues:

- The ability to change the content from one form to another.
- The ability to tie together different pieces of content processing.
- The ability to separate different documents for different types of processing.
- The ability to automate the entire procedure so documents can move through complex processing phases automatically.
- The ability to integrate manual steps or long-running, asynchronous operations in applications.

Flexibility is important in content processing, as both the starting points of documents and their end results can vary significantly. Also, application requirements can evolve over time, forcing the content processing application to change with the requirements. It is therefore necessary to have a content processing environment that can allow for such change.

MarkLogic Server provides capabilities to modify content in simple ways, such as with Information Studio transforms, and in more complex ways with workflows and pipelines. An example of a content processing application is The Default Conversion Option, which uses the components of the MarkLogic Content Processing Framework, and XQuery modules, to create a unified conversion process that converts Microsoft Office, Adobe PDF, and HTML files to well-structured XHTML and simplified DocBook format XML documents.

10.1 Converting Microsoft Office and Adobe PDF Into XML

The Default Conversion Option of the Content Processing Framework converts Microsoft Office, Adobe PDF, and HTML files to XHTML and DocBook. The Default Conversion Option only converts Microsoft Office 97 and newer documents; it cannot convert documents from Microsoft Office 95 or earlier.

10.2 Information Studio Transforms

MarkLogic Server Information Studio provides transforms for the following actions:

- Deleting Elements or Attributes
- Normalizing Dates
- Validating Documents Against a Schema
- Applying a Custom XSLT Stylesheet
- Extracting Metadata from Binary Content With the Filter Documents Transform
- Renaming Elements or Attributes

10.3 Converting to XHTML

MarkLogic provides facilities for converting documents to XHTML as follows:

- `xdmp:tidy` converts HTML to XHTML
- Default Conversion Option of the Content Processing Framework converts Microsoft Office, PDF, and HTML files to XHTML
- `xdmp:pdf-convert` converts a PDF file to XHTML
- `xdmp:excel-convert` converts a Microsoft Excel document to XHTML

10.4 Automating Metadata Extraction

MarkLogic provides facilities to extract and associate metadata from binary documents as follows:

- Filter Documents transform in Information Studio
- `xdmp:document-filter`, a built-in XQuery function
- MarkLogic content pump provides commands to include or exclude metadata during copy, export, and import

10.5 Transforming XML Structures

A common task sometimes required with XML is to transform one structure to another structure. A design pattern using the XQuery typeswitch expression to transform XML to XHTML or XSL-FO is described in Transforming XML Structures With a Recursive typeswitch Expression in the *Application Developer's Guide*.

11.0 Performance Considerations

This chapter covers the following topics:

- Understanding the Locking and Journaling Database Settings for Bulk Loads
- Fragmentation

11.1 Understanding the Locking and Journaling Database Settings for Bulk Loads

When you load content, MarkLogic Server performs updates transactionally, locking documents as needed and saving the content to disk in the journal before the transaction commits. By default, all documents are locked during an update and the journal is set to preserve committed transactions, even if the MarkLogic Server process ends unexpectedly.

The database settings `locking` and `journaling` control how fine-grained and robust you want this transactional process to behave. By default, it is set up to be a good balance of speed and data-integrity. All documents being loaded are locked, making it impossible for another transaction to update the same document being loaded or updated in a different transaction, and making it impossible to create duplicate URIs in your database.

There is a journal write to disk on transaction commit, and by default the system relies on the operating system to perform the disk write. Therefore, even if the MarkLogic Server process ends, the write to the journal occurs, unless the computer crashes before the operating system can perform the disk write. Protecting against the MarkLogic Server process ending unexpectedly is the `fast` setting for the `journaling` option. If you want to protect against the computer crashing unexpectedly, you can set the journaling to `strict`. A setting of `strict` forces a filesystem sync before the transaction is committed. This takes a little longer for each transaction, but protects your transactions against the computer failing.

If you are sure that no other programs are updating content in the database, and if you are sure that your program is not updating a URI more than one time, it is possible to turn the `journaling` and/or `locking` database settings to `off`. Turning these `off` might make sense, for example, during a bulk load. You should only do so if you are sure that no URIs are being updated more than once. Be sure to turn the `directory creation` database setting to `manual` before disabling `locking` in a database, as automatic directory creation creates directories if they do not already exist, and, without locking, can result in duplicate directory URIs in some cases. If you set the `locking` option to `fast`, it locks URIs for existing documents, but not for new documents. The default `locking` setting of `strict` locks for both new and existing documents, which is the safest option.

Warning Use extreme caution when changing these settings, as the `off` and `fast` settings disable and limit the transactional checks performed in the database, and doing so without understanding how it works can result in inconsistent data.

The advantage of disabling the `locking` or `journaling` settings is that it makes the loads faster. For bulk loads, where if something goes wrong you can simple start over, this might be a trade-off worth considering.

For more details on how transactions work, see Understanding Transactions in MarkLogic Server.

11.2 Fragmentation

Proper fragmentation is important to performance. Before you specify how to fragment the XML data being loaded, you need to plan your fragmentation strategy. For guidelines on fragmentation, see Choosing a Fragmentation Strategy in the *Administrator's Guide*.

12.0 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.