MarkLogic Server

REST Application Developer's Guide

MarkLogic 7
November, 2013

Last Revised: 7.0-4, September, 2014

Copyright © 2014 MarkLogic Corporation. All rights reserved.

Table of Contents

REST Application Developer's Guide

1.0 Introduction to the MarkLogic REST API 11
- 1.1 Capabilities of the MarkLogic REST API 11
- 1.2 Getting Started with the MarkLogic REST API 12
  - 1.2.1 Preparation 12
  - 1.2.2 Create a REST API Instance 13
  - 1.2.3 Load Documents Into the Database 14
  - 1.2.4 Search the Database 15
  - 1.2.5 Tear Down the REST API Instance 17
- 1.3 MarkLogic REST API Service Summary 18
- 1.4 Security Requirements 20
- 1.5 Terms and Definitions 21
- 1.6 Understanding REST Resources 22
  - 1.6.1 Addressing a Resource 22
  - 1.6.2 Specifying the REST API Version 23
  - 1.6.3 Specifying Parameters in a Resource Address 23
- 1.7 Understanding the Example Commands 23
  - 1.7.1 Introduction to the curl Tool 24
  - 1.7.2 Modifying the Example Commands for Windows 25
- 1.8 Controlling Input and Output Content Type 25
  - 1.8.1 General Content Type Guidelines 25
  - 1.8.2 Details on Content Type Determination 26
  - 1.8.3 Example: Inserting and Reading a Document 28
  - 1.8.4 Example: Inserting and Reading Metadata 28
  - 1.8.5 Example: Documents With No or Unknown URI Extension 29
  - 1.8.6 Example: Mixing Document and Non-Document Data 30
- 1.9 Working With JSON 31
- 1.10 Error Reporting 31

2.0 Administering MarkLogic REST API Instances 33
- 2.1 What is an Instance? 33
- 2.2 Creating an Instance 33
  - 2.2.1 Creating an Instance Interactively 34
  - 2.2.2 Creating an Instance Using HTTP 34
  - 2.2.3 Example: Creating an Instance Using HTTP 36
- 2.3 Removing an Instance 37
- 2.4 Retrieving Configuration Information 38
  - 2.4.1 Retrieving Configuration for All Instances 38
  - 2.4.2 Retrieving Instance Configuration by Content Database 39

2.4.3 Retrieving Instance Configuration by Instance Name 40
2.5 Configuring Instance Properties 41
2.5.1 Instance Configuration Properties 42
2.5.2 Listing Instance Property Settings 42
2.5.3 Setting Instance Properties 44
2.5.4 Resetting Instance Properties 45

3.0 Manipulating Documents 46
3.1 Summary of Document Management Services 46
3.1.1 Summary of the /documents Service 47
3.1.2 Summary of the /graphs Service 48
3.2 Loading Content into the Database 48
3.2.1 Loading Content 49
3.2.2 Adding Metadata 50
3.2.2.1 Inserting or Updating Metadata 50
3.2.2.2 Example: Replacing One Metadata Category Using XML 51
3.2.2.3 Example: Replacing Multiple Metadata Categories Using XML 51
3.2.2.4 Example: Replacing Multiple Metadata Categories Using JSON 52
3.2.3 Loading Content and Adding Metadata in the Same Request 52
3.2.3.1 Loading Content and Metadata Using Request Parameters 52
3.2.3.2 Loading Content and Metadata Using a Multipart Message 53
3.2.4 Automatically Generating Document URIs 55
3.2.5 Loading JSON Documents Into the Database 56
3.2.6 Loading Triples 56
3.2.7 Controlling Access to Documents Created with the REST API 57
3.2.8 Transforming Content During Ingestion 58
3.3 Retrieving Documents from the Database 58
3.3.1 Retrieving the Contents of a Document 58
3.3.2 Retrieving Metadata About a Document 59
3.3.3 Retrieving Content and Metadata in a Single Request 59
3.3.4 Transforming Content During Retrieval 61
3.4 Partially Updating Document Content or Metadata 61
3.4.1 Introduction to Content and Metadata Patching 62
3.4.2 Basic Steps for Patching Documents 63
3.4.3 XML Patch Reference 64
3.4.3.1 patch 64
3.4.3.2 insert 65
3.4.3.3 replace 67
3.4.3.4 replace-insert 69
3.4.3.5 delete 72
3.4.3.6 replace-library 73
3.4.4 Managing XML Namespaces in a Patch 74
3.4.5 XML Examples of Partial Updates 75
3.4.5.1 Example cURL Commands 75

3.4.5.2 Example: Inserting an Element 76
3.4.5.3 Example: Inserting an Attribute 76
3.4.5.4 Example: Inserting a Text Node 77
3.4.5.5 Example: Inserting a Comment or Processing Instruction 77
3.4.5.6 Example: Multiple Inserts in One Patch 78
3.4.5.7 Example: Replacing an Element or Element Contents 79
3.4.5.8 Example: Replacing an Attribute Value 79
3.4.5.9 Example: Replacing or Inserting an Element in One Operation ... 80
3.4.5.10 Example: Replacing or Inserting an Attribute in One Operation .. 81
3.4.5.11 Example: Deleting an Element or Attribute 82
3.4.6 JSON Patch Reference 82
3.4.6.1 patch 83
3.4.6.2 insert 84
3.4.6.3 replace 85
3.4.6.4 replace-insert 87
3.4.6.5 delete 90
3.4.6.6 replace-library 91
3.4.7 JSON Examples of Partial Update 92
3.4.7.1 Example cURL Commands 92
3.4.7.2 Example: Insert 93
3.4.7.3 Example: Replace 94
3.4.7.4 Example: Replace-Insert 94
3.4.7.5 Example: Delete 96
3.4.8 Patching Metadata 98
3.4.9 Constructing Replacement Data on the Server 100
3.4.9.1 Using a Replacement Constructor Function 100
3.4.9.2 Using Builtin Replacement Constructors 103
3.4.9.3 Writing a User-Defined Replacement Constructor 104
3.4.9.4 Installing or Updating a User-Defined Replace Library 105
3.4.10 How Position Affects the Insertion Point 106
3.4.10.1 Specifying Position in XML 106
3.4.10.2 Specifying Position in JSON 108
3.4.11 Path Expressions Usable in Index Definitions 109
3.4.12 Introduction to JSONPath 110
3.5 Performing a Lightweight Document Check 111
3.6 Removing Documents from the Database 111
3.6.1 Removing a Document or Metadata 112
3.6.2 Removing Multiple Documents 112
3.6.3 Removing All Documents 113
3.6.4 Removing a Semantic Graph 113
3.7 Using Optimistic Locking to Update Documents 114
3.7.1 Understanding Optimistic Locking 114
3.7.2 Enabling Optimistic Locking 115
3.7.3 Obtaining a Version Id 116

3.7.4 Applying a Conditional Update 117
3.7.5 Example: Updating a Document Using Optimistic Locking 118
3.8 Client-Side Cache Management Using Content Versioning 120
3.8.1 Enabling Content Versioning 120
3.8.2 Using Content Versioning for Cache Refreshing 121
3.8.3 Example: Refreshing a Cached Document 121
3.9 Working with Binary Documents 123
3.9.1 Types of Binary Documents 123
3.9.2 Streaming Binary Content 123
3.10 Working with Metadata 124
3.10.1 Metadata Categories 124
3.10.2 XML Metadata Format 125
3.10.3 JSON Metadata Format 126
3.10.4 Working With Properties Using JSON 127
3.10.5 Disabling Metadata Merging 127
3.10.5.1 When to Consider Disabling Metadata Merging 128
3.10.5.2 Understanding Metadata Merging 128
3.10.5.3 How to Disable Metadata Merging 128

4.0 Using and Configuring Query Features 130
4.1 Query Feature Overview 130
4.2 Querying Documents and Metadata 132
4.2.1 Constraining a Query by Collection or Directory 132
4.2.2 Searching With String Queries 133
4.2.3 Searching With Structured Queries 133
4.2.4 Searching By JSON Key and Value 134
4.2.5 Searching By Element or Attribute Name and Value 136
4.2.6 Debugging /search Queries With Logging 137
4.3 Querying Lexicons and Range Indexes 140
4.3.1 Querying the Values in a Lexicon or Range Index 140
4.3.2 Finding Value Co-Occurrences in Lexicons 143
4.3.3 Using a Query to Constrain Results 146
4.3.4 Identifying Lexicon and Range Index Values in Query Options 147
4.3.4.1 Defining Queryable Lexicon or Range Index Values 147
4.3.4.2 Defining Queryable Lexicon or Range Index Co-Occurrences 149
4.3.5 Creating Indexes on JSON Keys 150
4.3.6 Limiting the Number of Results 151
4.4 Using Query By Example to Prototype a Query 152
4.4.1 What is QBE 152
4.4.2 Searching Documents With QBE 153
4.4.3 Validating a QBE 154
4.4.4 Generating a Combined Query from a QBE 155
4.5 Analyzing Lexicons and Range Indexes With Aggregate Functions 156
4.5.1 Aggregate Function Overview 156
4.5.2 Using Query Options to Apply Aggregate Functions 157

4.5.3 Using Request Parameters to Apply Aggregate Functions 158
4.5.4 Example: Applying a Builtin Aggregate Function 160
4.5.5 Example: Applying an Aggregate UDF 160
4.6 Specifying Dynamic Query Options with Combined Query 161
4.6.1 Syntax 162
4.6.2 Interaction with Queries in Request Parameters 162
4.6.3 Interaction with Persistent Query Options 163
4.6.4 Performance Considerations 164
4.6.5 Combined Query Examples 165
4.6.5.1 Example: Overriding Persistent Constraints 165
4.6.5.2 Example: Modifying the Search Response 166
4.7 Querying Triples 166
4.8 Configuring Query Options 167
4.8.1 Controlling Queries With Options 167
4.8.2 Adding Query Options to a Request 168
4.8.3 Creating or Modifying Query Options 169
4.8.4 Creating or Modifying One Option 171
4.8.5 Checking Index Availability 172
4.8.6 Retrieving Options 173
4.8.7 Retrieving a List of Installed Query Options 174
4.8.8 Removing Query Options 175
4.8.8.1 Removing Query Options 176
4.8.8.2 Removing a Single Option 176
4.8.8.3 Removing All Named Query Options 176
4.9 Using Namespace Bindings 177
4.9.1 When Do You Need a Namespace Binding 177
4.9.2 Creating or Updating a Namespace Binding 177
4.9.3 Creating or Updating Multiple Namespace Bindings 179
4.9.4 Listing Available Namespace Bindings 181
4.9.5 Deleting Namespace Bindings 183
4.10 Generating Search Facets 183
4.11 Paginating Results 185
4.12 Customizing Search Results 186
4.12.1 Customizing Search Snippets 187
4.12.1.1 Customizing the Default Search Snippets 187
4.12.1.2 Creating Your Own Snippet Extension 191
4.12.1.3 Generating Custom JSON Snippets 196
4.12.2 Transforming the Search Response 198
4.13 Generating Search Term Completion Suggestions 199
4.13.1 Basic Steps 199
4.13.2 Example: Generating Search Suggestions Using GET 200
4.13.2.1 Initialize the Database 201
4.13.2.2 Install Query Options 202
4.13.2.3 Retrieve Unconstrained Search Suggestions 202
4.13.2.4 Retrieve Constrained Search Suggestions 203
4.13.3 Example: Generating Search Suggestions Using POST 203

4.13.4 Where to Find More Information 205

5.0 Reading and Writing Multiple Documents 206
5.1 Terms and Definitions 206
5.2 Writing Multiple Documents 207
5.2.1 Example: Loading Multiple Documents 208
5.2.2 Request Body Overview 210
5.2.3 Response Overview 211
5.2.4 Constructing a Content Part 213
5.2.4.1 Controlling Document Type 214
5.2.4.2 Specifying an Explicit Document URI 214
5.2.4.3 Automatically Generating a Document URI 215
5.2.4.4 Adding Content Options 215
5.2.4.5 Example Content Part Headers 215
5.2.5 Understanding Metadata Scoping 216
5.2.6 Understanding When Metadata is Preserved or Replaced 219
5.2.7 Constructing a Metadata Part 220
5.2.7.1 Constructing a Request Default Metadata Part 220
5.2.7.2 Constructing a Document-Specific Metadata Part 221
5.2.8 Applying a Write Transformation 225
5.2.9 Example: Controlling Metadata Through Defaults 225
5.2.9.1 Payload Description 225
5.2.9.2 Generating the POST Body 227
5.2.9.3 Executing the Request 228
5.2.9.4 Verifying the Results 229
5.2.10 Example: Adding Documents to a Collection 230
5.2.10.1 Payload Description 231
5.2.10.2 Generating the POST Body 231
5.2.10.3 Executing the Request 233
5.2.10.4 Verifying the Results 234
5.2.10.5 Extending the Example 236
5.2.11 Generating Example Payloads with XQuery 236
5.3 Reading Multiple Documents by URI 239
5.4 Reading Multiple Documents Matching a Query 240
5.4.1 Overview of Bulk Read By Query 240
5.4.2 Example: Using Query By Example (QBE) 241
5.4.3 Example: Using a String, Structured, or Combined Query 243
5.4.4 Including Search Results in the Response 245
5.4.5 Paginating Results 246
5.5 Bulk Read Response Overview 247

6.0 Managing Transactions 250
6.1 Service Summary 250
6.2 Overview of RESTful Transactions 251
6.3 Creating a Transaction 252

6.4 Associating a Transaction with a Request 253
6.5 Committing or Rolling Back a Transaction 253
6.6 Checking Transaction Status 253
6.7 Managing Transactions When Using a Load Balancer 255

7.0 Alerting 257
7.1 Summary of /alert Services 257
7.2 Alerting Pre-Requisites 257
7.3 Alerting Concepts 258
7.4 Defining an Alerting Rule 259
7.5 Installing an Alerting Rule 260
7.6 Testing for Matches to Alerting Rules 261
7.6.1 Basic Steps 261
7.6.2 Identifying Input Documents Using a Query 263
7.6.3 Identifying Input Documents Using URIs 264
7.6.4 Matching Against a Transient Document 264
7.6.5 Filtering Match Results 265
7.6.6 Transforming Match Results 265
7.6.6.1 Writing a Match Result Transform 265
7.6.6.2 Using a Match Result Transform 266
7.7 Retrieving Rule Definitions 267
7.8 Testing for the Existence of Rule 267
7.9 Deleting a Rule 268

8.0 Working With Content Transformations 269
8.1 Writing Transformations 269
8.1.1 Guidelines for Writing Transforms 269
8.1.2 Writing XQuery Transformations 270
8.1.3 Writing XSLT Transformations 272
8.1.4 Expected Input and Output 273
8.1.5 Reporting Errors 274
8.1.6 Context Map Keys 276
8.1.7 Controlling Transaction Mode 276
8.2 Installing Transformations 277
8.2.1 Basic Installation 277
8.2.2 Defining Transform-Specific Parameters 278
8.2.3 Including Transform Metadata 278
8.3 Applying Transformations 278
8.4 Discovering Transformations 279
8.5 Retrieving the Implementation of a Transformation 280
8.6 XQuery Example: Adding an Attribute During Ingestion 280
8.7 XSLT Example: Adding an Attribute During Ingestion 282

9.0 Extending the REST API 285
9.1 Understanding Resource Service Extensions 285

9.2 Creating a Resource Service Extension 285
9.2.1 Guidelines for Writing Resource Service Extensions 286
9.2.2 The Resource Extension Interface 286
9.2.3 Returning JSON Documents 288
9.2.4 Reporting Errors 288
9.2.5 Setting the Response Status Code 288
9.2.6 Setting Additional Response Headers 289
9.2.7 Context Map Keys 290
9.2.8 Controlling Transaction Mode 291
9.3 Installing a Resource Service Extension 291
9.4 Using a Resource Service Extension 293
9.5 Controlling Access to a Resource Service Extension 293
9.6 Discovering Resource Service Extensions 293
9.7 Retrieving the Implementation of a Resource Service Extension 294
9.8 Example Resource Service Extension 295
9.8.1 Overview 295
9.8.2 Installing the Example Extension 296
9.8.3 Using the Example Extension 297
9.8.4 Example Extension Implementation 298
9.9 Managing Dependent Libraries and Other Assets 299
9.9.1 Maintenance of Dependent Libraries and Other Assets 300
9.9.2 Installing or Updating an Asset 300
9.9.3 Referencing an Asset 301
9.9.4 Removing an Asset 301
9.9.5 Retrieving an Asset List 302
9.9.6 Retrieving an Asset 303

10.0 Migrating REST Applications 304
10.1 Before You Begin 304
10.2 Migrating the REST API Instance and Database Configuration 304
10.2.1 Export the Configuration from the Source Cluster 305
10.2.2 Import the Configuration to the Destination Cluster 305
10.3 Migrating the Contents of the Database 306

11.0 Appendix A: JSON Query Options Reference 308
11.1 How to Use This Reference 309
11.2 Query Options 309
11.3 attribute-descriptor 315
11.4 bucket 315
11.5 collection-constraint 317
11.6 computed-bucket 317
11.7 constraint 319
11.8 container-constraint 321
11.9 custom-constraint 322
11.10 default-suggestion-source 324

11.11 element-descriptor 325
11.12 extract-metadata 326
11.13 field-descriptor 328
11.14 geo-attr-pair-constraint 328
11.15 geo-elem-constraint 330
11.16 geo-elem-pair-constraint 331
11.17 geo-path-constraint 332
11.18 grammar 333
11.19 heatmap-descriptor 335
11.20 joiner 335
11.21 operator 337
11.22 path-expression 339
11.23 properties-constraint 340
11.24 range-constraint 340
11.25 result-decorator 343
11.26 sort-order 344
11.27 starter 346
11.28 suggestion-source 348
11.29 term-definition 350
11.30 transform-results 353
11.31 value-constraint 355
11.32 values-or-tuples 356
11.33 word-constraint 358
11.34 word-lexicon 359
11.35 facet-options 360
11.36 term-options 361
11.37 values-options 362
11.38 geo-options 363
11.39 range-option 364
11.40 suggestion-option 365

12.0 Appendix B: XML Query Options Reference 366
12.1 <options> 366
12.2 <additional-query> 367
12.3 <concurrency-level> 367
12.4 <constraint> 367
12.4.1 <value> 368
12.4.2 <word> 369
12.4.3 <collection> 369
12.4.4 <range> 369
12.4.5 <container> 371
12.4.6 <element-query> 371
12.4.7 <properties> 372
12.4.8 <geo-*> 372
12.4.9 <custom> 373
12.4.10<term-option> 374

12.4.11<facet-option> 375
12.4.12<range-option> 376
12.4.13<geo-option> 376
12.5 <debug> 377
12.6 <default-suggestion-source> 377
12.6.1 <collection> 378
12.6.2 <range> 378
12.6.3 <word> 379
12.6.4 <word-lexicon> 379
12.7 <extract-metadata> 379
12.8 <forest> 379
12.9 <fragment-scope> 380
12.10 <grammar> 380
12.10.1<implicit> 381
12.10.2<joiner> 381
12.10.3<starter> 382
12.11 <operator> 384
12.12 <result-decorator> 384
12.13 <page-length> 385
12.14 <quality-weight> 385
12.15 <return-constraints> 385
12.16 <return-facets> 385
12.17 <return-metrics> 385
12.18 <return-plan> 385
12.19 <return-qtext> 386
12.20 <return-query> 386
12.21 <return-results> 386
12.22 <return-similar> 386
12.23 <search-option> 386
12.24 <searchable-expression> 386
12.25 <sort-order> 387
12.26 <suggestion-source> 387
12.27 <term> 389
12.28 <transform-results> 391
12.29 <tuples> 392
12.30 <values> 392

13.0 Technical Support 394

Combined Product Notices 395
COPYRIGHT 395
TRADEMARK NOTICE 472

1.0 Introduction to the MarkLogic REST API

The MarkLogic REST API provides a set of RESTful services for creating, updating, retrieving, deleting and query documents and metadata. This section provides a brief overview of the features of the API.

- Capabilities of the MarkLogic REST API
- Getting Started with the MarkLogic REST API
- MarkLogic REST API Service Summary
- Security Requirements
- Terms and Definitions
- Understanding REST Resources
- Understanding the Example Commands
- Controlling Input and Output Content Type
- Working With JSON
- Error Reporting

1.1 Capabilities of the MarkLogic REST API

The MarkLogic REST API is a RESTful interface for building client applications. The capabilities of the API include the following:

- Create, retrieve, update, and delete documents, metadata, and semantic triples in a MarkLogic database.
- Search documents and semantic graphs and query lexicon values, using several query formats, including string query, structured query, combined query, Query by Example (QBE), and SPARQL.
- Customize your queries by configuring dynamic and/or persistent query options.
- Apply transformations to document contents and search results.
- Extend the API to expose custom capabilities you author in XQuery and install on MarkLogic Server.

For a complete list of services, see “MarkLogic REST API Service Summary” on page 18.

You can use the MarkLogic REST API to work with XML, JSON, text, and binary documents. In most cases, your application can use either XML or JSON to exchange non-document data such as queries and search results with MarkLogic Server. For details, see:

- “Controlling Input and Output Content Type” on page 25
- “Working With JSON” on page 31

REST API client applications interact with MarkLogic Server through a REST API instance, a specially configured HTTP App Server. Each REST API instance is intended to service a single content database and client application. You can create and configure a REST API instance via REST request or interactively. For details, see "Administering MarkLogic REST API Instances" on page 33.

You can configure whether errors are returned to your application as XML or JSON. For details, see "Error Reporting" on page 31.

1.2 Getting Started with the MarkLogic REST API

This section leads you through a simple example that uses the MarkLogic REST API to insert documents into a database and perform a search. We will follow these steps:

1. Create a REST API Instance
2. Load Documents Into the Database
3. Search the Database
4. Tear Down the REST API Instance

1.2.1 Preparation

Before beginning this walkthrough, you should have the following software installed:

- MarkLogic Server, version 6.0-1 or later
- `curl`, a command line tool for issuing HTTP requests, or an equivalent tool.

Though the examples rely on `curl`, you can use any tool capable of sending HTTP requests. If you are not familiar with `curl` or do not have `curl` on your system, see "Introduction to the curl Tool" on page 24.

To create the input documents used by the walkthrough:

1. Create a text file named `one.xml` with the following contents:

```
<one>
  <child>The noble Brutus has told Caesar was ambitious</child>
</one>
```

2. Create a text file named `two.json` with the following contents:

```
{
  "two": {
    "child": "I come to bury Caesar, not to praise him."
  }
}
```

1.2.2 Create a REST API Instance

You must have a REST API *instance* to use the MarkLogic REST API. A REST API instance is an HTTP App Server specially configured to service HTTP requests against the API. For details, see “What is an Instance?” on page 33.

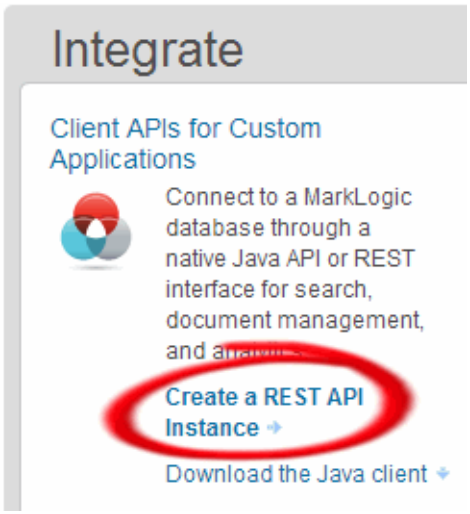
> **Note:** Each REST API instance can host a single application. You cannot share the modules database across multiple REST API instances.

This section walks you through creating a REST API instance interactively from the Welcome page. You can also create an instance using an HTTP request or Information Studio. For details, see “Creating an Instance” on page 33.

1. Use your browser to navigate to the Application Services Welcome page on port 8000. For example, navigate to the following URL if MarkLogic Server is on your local host.

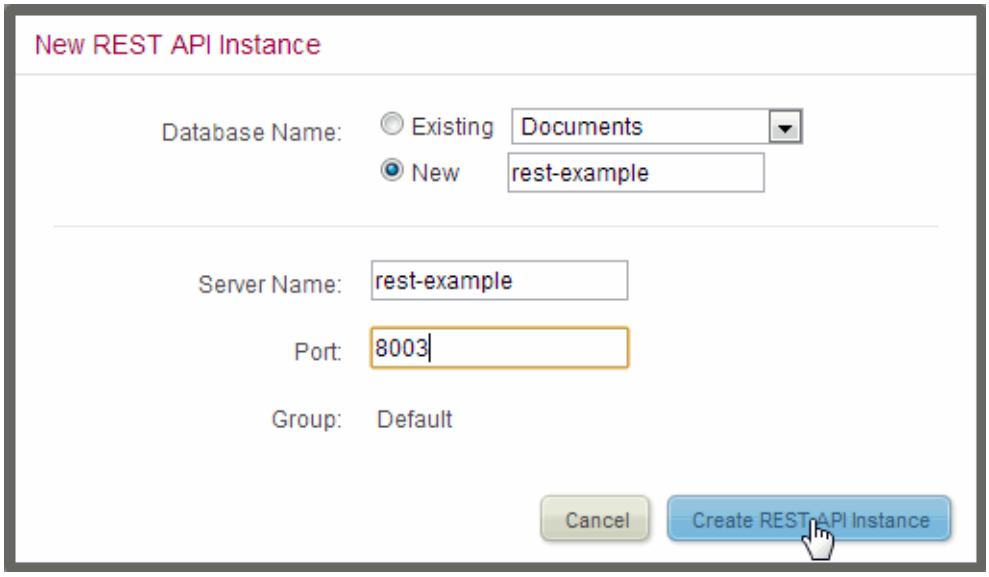
   ```
   http://localhost:8000
   ```

2. Locate the “Client APIs for Custom Applications” topic and click on the “Create a REST API Instance” link. The “New REST API Instance” dialog appears.



3. Under Database Name, click on the New radio button and enter `rest-example` as the database name.

4. Under Server Name, enter `rest-example`. The dialog should look similar to the following:



5. Click Create REST API Instance. Your instance, database, and forests are created. The confirmation dialog appears.

6. Click Done on the confirmation dialog to dismiss it.

7. Navigate to the Admin Interface in your browser:

```
http://localhost:8001
```

   Observe the existence of a database named `rest-example` with a forest named `rest-example-1`, an HTTP App Server called `rest-example` attached to this database, and a modules database named `rest-example-modules` associated with the App Server.

8. Test the instance by navigating to the following URL in your browser. (Your port number may differ if port 8003 was already in use). When prompted for a username and password, authenticate with a user that has the `rest-reader` role.

```
http://localhost:8003
```

   If the instance is properly configured, you will see a short HTML summary of capabilities provided by the REST API.

1.2.3 Load Documents Into the Database

This procedure loads sample content into the "rest-example" database using the `/documents` service. The `/documents` service allows you to create, read, update and delete documents in the database.

To load the sample documents into the `rest-example` database:

1. Navigate to the directory containing the sample documents you created in "Preparation" on page 12.

2. Execute the following command to load `one.xml` into the database with the URI `/xml/one.xml`:

```
$ curl --anyauth --user user:password -X PUT -d@'./one.xml' \
    -H "Content-type: application/xml" \
    'http://localhost:8003/v1/documents?uri=/xml/one.xml'
```

   The URL tells the `/documents` service to create an XML document with database URI `/xml/one.xml` (`uri=...`) from the contents in the request body. If the request succeeds, the service returns status 201 (Document Created).

3. Execute the following command to load `two.json` into the database with the URI `/json/two.json`:

```
$ curl --anyauth --user user:password -X PUT -d@'./two.json' \
  -H "Content-type: application/json" \
  'http://localhost:8003/v1/documents?uri=/json/two.json'
```

4. Optionally, use Query Console to explore the `rest-example` database. The database should contain 2 documents, `/xml/one.xml` and `/json/two.json`.

To learn more about the document manipulation features of the `documents` service, see "Manipulating Documents" on page 46.

1.2.4 Search the Database

The MarkLogic REST API provides several query services. This procedure uses the `search` service to perform a simple string query search of the "rest-examples" database, finding documents containing "caesar". For details, see "Using and Configuring Query Features" on page 130.

To search the database:

1. Execute the following command to send a search request to the instance, requesting matches to the search string "caesar". Results are returned as XML by default.

```
$ curl --anyauth --user user:password \
    'http://localhost:8003/v1/search?q=caesar'
```

2. Examine the XML search results returned in the response body. Notice that there are two matches, one in each document.

```
<search:response snippet-format="snippet" total="2" start="1" ...>
  <search:result index="1" uri="/xml/one.xml" ...>
    <search:snippet>
      <search:match
path="fn:doc(&quot;/xml/one.xml&quot;)/one/child">The noble Brutus has
told <search:highlight>Caesar</search:highlight> was
ambitious</search:match>
    </search:snippet>
  </search:result>
  <search:result index="2" uri="/json/two.json"
path="fn:doc(&quot;/json/two.json&quot;)" score="2048"
confidence="0.283107" fitness="0.235702">
    <search:snippet>
      <search:match
path="fn:doc(&quot;/json/two.json&quot;)/*:json/*:two/*:child">I come
to bury <search:highlight>Caesar</search:highlight>, not to praise
him.</search:match>
    </search:snippet>
  </search:result>
  <search:qtext>caesar</search:qtext>
  <search:metrics>...</search:metrics>
</search:response>
```

3. Run the search command again, generating JSON output by setting the Content-type header to `application/json`:

```
$ curl --anyauth --user user:password \
    -H "Content-type: application/json" \
    'http://localhost:8003/v1/search?q=caesar'

{
  "snippet-format": "snippet",
  "total": 2,
  "start": 1,
  "page-length": 10,
  "results": [
    {
      "index": 1,
      "uri": "\/xml\/one.xml",
      "path": "fn:doc(\"\/xml\/one.xml\")",
      "score": 2048,
      "confidence": 0.283107,
      "fitness": 0.235702,
      "matches": [
        {
          "path": "fn:doc(\"\/xml\/one.xml\")\/one\/child",
          "match-text": [
            "The noble Brutus has told ",
            {
              "highlight": "Caesar"
            },
            " was ambitious"
          ]
```

```
          }
        ]
      },
      {
        "index": 2,
        "uri": "\/json\/two.json",
        "path": "fn:doc(\"\/json\/two.json\")",
        "score": 2048,
        "confidence": 0.283107,
        "fitness": 0.235702,
        "matches": [
          {
            "path":
"fn:doc(\"\/json\/two.json\")\/*:json\/*:two\/*:child",
            "match-text": [
              "I come to bury ",
              {
                "highlight": "Caesar"
              },
              ", not to praise him."
            ]
          }
        ]
      }
    ],
    "qtext": "caesar",
    "metrics": { ... }
}
```

Additional query features allow you to search using structured queries or Query By Example (QBE), to search by JSON key and value or element and element attribute values, and to search and analyze lexicons and range indexes. You can also define search options to tailor your search and results. For details, see “Using and Configuring Query Features” on page 130.

1.2.5 Tear Down the REST API Instance

This procedure uses the `rest-apis` service on port 8002 to remove the MarkLogic REST API instance created in “Create a REST API Instance” on page 13. By default, removing the instance leaves the content and modules databases associated with the instance intact, but in this example we remove them by using the `include` request parameter.

Note: Tearing down a REST API instance causes a server restart.

Follow this procedure to remove the “rest-examples” instance and associated content and modules databases:

1. Execute the following shell command to remove the instance and database:

```
$ curl --anyauth --user user:password -X DELETE \
  'http://localhost:8002/v1/rest-apis/rest-example?include=content&inc
lude=modules'
```

2. Navigate to the Admin Interface in your browser to confirm removal of the databases and App Server:

```
http://localhost:8001
```

1.3 MarkLogic REST API Service Summary

The table below gives a brief overview of the services provided by the MarkLogic REST API and where to find out more information. Additional, finer grained services are available in many cases. For example, in addition to `/config/query` (manage query options), there is a `/config/query/{name}` service (manage a specific option). For details, refer to the *MarkLogic REST API Reference*.

These services are made available through an HTTP App Server when you create an instance of the MarkLogic REST API. For details, see "Creating an Instance" on page 33.

Service	Description	More Information
`/rest-apis`	MarkLogic REST API instance administration, including creating and tearing down instances.	"Administering MarkLogic REST API Instances" on page 33
`/documents`	Document manipulation, including creating, updating and deleting documents and meta data.	"Manipulating Documents" on page 46
`/search`	Search content and metadata using string and structured queries.	"Using and Configuring Query Features" on page 130
`/qbe`	Search content using Query By Example, a query syntax that closely resembles the structure of your documents.	"Using Query By Example to Prototype a Query" on page 152
`/keyvalue`	Search by XML content by element, element attribute, and field name and value. Search JSON content by JSON key and value.	"Using and Configuring Query Features" on page 130
`/values`	Retrieve lexicon and range index values and value co-occurrences. Apply builtin and user-defined aggregate functions to lexicon and range index values and value co-occurrences.	"Using and Configuring Query Features" on page 130

Service	Description	More Information
`/suggest`	Retrieve text completion suggestions based on query text entered by the user.	"Generating Search Term Completion Suggestions" on page 199
`/graphs`	Store and manage semantic triples data.	"Loading Triples" on page 56
`/graphs/sparql`	Performan semantic queries using SPARQL.	"Querying Triples" on page 166
`/transactions`	Support for evaluating REST requests in multi-statement transactions. Create, commit, rollback, and monitor transactions.	"Managing Transactions" on page 250
`/alert`	Support for creating alerting applications.	"Alerting" on page 257
`/resources`	Access to user-defined resource service extensions.	"Extending the REST API" on page 285
`/config/query`	Create, modify, delete, and read configuration options used to control queries made using /search, /keyvalue, and /values.	"Configuring Query Options" on page 167
`/config/indexes`	Compare query options against the database configuration to determine whether all required indexes are configured in the database.	"Checking Index Availability" on page 172
`/config/properties`	Configure instance-wide properties, such as enabling debug output and setting the content type of error messages.	"Configuring Instance Properties" on page 41
`/config/transforms`	Create, update, delete, and read user-defined content transformations. Transformations can be used to modify content when it is inserted into or retrieved from the database using the `/documents` service, or to transform search results.	"Working With Content Transformations" on page 269
`/config/resources`	Manage resource service extensions.	"Extending the REST API" on page 285

Service	Description	More Information
`/ext`	Manage assets in the modules database associated with a REST API instance, such as dependent XQuery library modules used by transformations and resource service extensions.	"Managing Dependent Libraries and Other Assets" on page 299
`/config/namespaces`	Create and manage instance-wide namespace prefix bindings. Such bindings allow you to use namespace prefixes in queries.	"Using Namespace Bindings" on page 177

1.4 Security Requirements

The user with which you make a MarkLogic REST API request must have appropriate privileges for the content accessed by the request, such as permission to read or update documents in the target database.

In addition, the user must use one of the pre-defined roles listed below, or the equivalent privleges. The role/privilege requirement for each MarkLogic REST API operation is listed in the *MarkLogic REST API Reference*.

Role	Description
`rest-reader`	Enables read operations through the MarkLogic REST API, such as retrieving documents and metadata. This role does not grant any other privileges, so the user might still require additional privileges to read content.
`rest-writer`	Enables write operations through the MarkLogic REST API, such as creating documents, metadata, or configuration information. This role does not grant any other privileges, so the user might still require additional privileges to write content.
`rest-admin`	Enables administrative operations through the MarkLogic REST API, such as creating an instance and managing instance configuration. This role does not grant any other privileges, so the user might still require additional privileges.

Documents you create with the MarkLogic REST API have a read permission for the `rest-reader` role and an update permission for the `rest-writer` role. To restrict access to particular users, create custom roles rather than assigning users to the default `rest-*` roles. For details, see "Controlling Access to Documents Created with the REST API" on page 57.

For details about roles and privileges, see the *Understanding and Using Security Guide*.

1.5 Terms and Definitions

The following terms and definitions are used in this guide:

Term	Definition
REST	REpresentational State Transfer, an architecture style that, in the context of the MarkLogic REST API, describes the use of HTTP to make calls between a client application and MarkLogic Server to create, update, delete and query content and metadata in the database.
resource	An abstraction of a MarkLogic REST API service, as presented by the REST architecture.
resource address	A URL that identifies a MarkLogic Server resource. Resource addresses are described in “Understanding REST Resources” on page 22.
rewriter	An XQuery module that interprets the URL of an incoming HTTP request and rewrites it to an internal URL that services the request.
REST API instance	An instantiation of the MarkLogic REST API against which applications can make RESTful HTTP requests. An instance consists of an HTTP App Server, a URL rewriter, a content database, a modules database, and the modules that implement the API. For details, see “Administering MarkLogic REST API Instances” on page 33.
extension	An user-defined XQuery module that implements additional resource services that are made available through the MarkLogic REST API. For details, see “Extending the REST API” on page 285.

Term	Definition
string query	A simple search string constructed using either the default MarkLogic Server search grammar, or a user-defined grammar. For example, “cat” and “cat OR dog” are string queries. For details, see “Querying Documents and Metadata” on page 132.
structured query	The pre-parsed representation of a query, expressed as XML or JSON. Structured queries allow you to express complex queries very efficiently. For details, see “Querying Documents and Metadata” on page 132 and Searching Using Structured Queries in the *Search Developer’s Guide*.
lexicon	A list of unique words or values, either throughout an entire database or within named elements, attributes, or fields. You can also define lexicons that allow quick access to the document and collection URIs in the database. Lexicons are usually backed by a range index. For details, see “Querying Lexicons and Range Indexes” on page 140 and Browsing With Lexicons in the *Search Developer’s Guide*.
endpoint	An XQuery module on MarkLogic Server that is invoked by and responds to an HTTP request for monitoring information.

1.6 Understanding REST Resources

This section covers the basic structure of a MarkLogic REST API URL. If you are already familiar with REST resource addressing, you can skip this section. The following topics are covered:

- Addressing a Resource
- Specifying the REST API Version
- Specifying Parameters in a Resource Address

1.6.1 Addressing a Resource

A resource address takes the form of a URL that includes a host name and a port number:

```
http://host:port/version/resource/
```

The host and port must reference a host running MarkLogic Server with a MarkLogic REST API instance running on that port. A MarkLogic REST API instance is served by an HTTP App Server. For details, see “Creating an Instance” on page 33.

A resource address always includes the API version in URL. For details, see “Specifying the REST API Version” on page 23.

You can optionally include parameters in a resource address as follows:

```
http://host:port/version/resource?param=value&param=value
```

For details, see “Specifying Parameters in a Resource Address” on page 23.

1.6.2 Specifying the REST API Version

To guarantee stable behavior of the MarkLogic REST API as new versions are released, each resource address in the MarkLogic REST API includes a version number. The examples in this chapter show the version as `v1` or simply *version*. The version has the format:

```
v#
```

Where # is the version number. For example, in the initial version of the API:

```
http://localhost:8003/v1/documents
```

The current version number is `v1`.

Note: The version number is only updated when resource addresses and/or parameters have changed. It is not updated when resource addresses and/or parameters are added or removed.

1.6.3 Specifying Parameters in a Resource Address

Resource addresses accept request parameters to tailor behavior or control input and output format.

To specify multiple parameters, use the ‘?’ sign before the first parameter and the ‘&’ sign before any additional parameters:

```
http://host:port/version/resource?param1=value&param2=value....
```

See the *MarkLogic REST API Reference* for a list of parameters available with each resource.

1.7 Understanding the Example Commands

The examples in this guide use the curl command line tool to send HTTP requests that exercise the MarkLogic REST API. The examples also use Unix command line syntax. Review this section if you are not familiar with curl or Unix command line syntax. If you are not familiar with RESTful URL conventions, see “Understanding REST Resources” on page 22.

- Introduction to the curl Tool
- Modifying the Example Commands for Windows

1.7.1 Introduction to the curl Tool

`curl` is a command line tool for sending HTTP requests. If you do not have `curl`, you can download a copy from http://curl.haxx.se/download.html, or use an equivalent tool. This section provides a brief overview of the `curl` command line options used in this guide. For details, see the `curl` man page or the online documentation at http://curl.haxx.se/docs/.

The `curl` command line is of the form:

```
curl options URL
```

The options most often used in the examples in this guide are summarized in the table below.

Option	Description
--anyauth	Have `curl` figure out the authentication method. The method depends on your REST API instance App Server configuration. You can specify an explicit method using options such as `--digest` or `--basic`.
--user *username:password*	Username and password with which to authenticate the request. Use a MarkLogic Server user that has sufficient privileges to carry out the requested operation. For details, see “Security Requirements” on page 20.
-X *http_method*	The type of HTTP request (GET, PUT, POST, DELETE) that `curl` should send. If `-X` is not given, GET is assumed.
-d *data*	Data to include in the request body. Data may be placed directly on the command line as an argument to `-d`, or read from a file by using `@filename`. The examples in this guide usually read from file to simplify the command line. For example, `curl -X POST -d @./my-body.xml ...` reads the post body contents from the file ./my-body.xml.
-H *headers*	HTTP headers to include in the request. This is most often used by the examples to specify `Accept` and `Content-type headers`.
-i	Specifies that the curl output should include the HTTP response headers in the output. By default, curl doesn’t display the response header.

For example, the following command sends a POST request with the contents of the file `my-body.json` in the request body, and specifies the Content-type as `application/json`:

```
$ curl --anyauth --user me:mypassword -X POST -d @my-body.json \
    -H "Content-type: application/json" \
    http://localhost:8003/v1/config/query/my-options
```

1.7.2 Modifying the Example Commands for Windows

The command line examples in this guide use Unix command line syntax, usable from either a Unix or Cygwin command line. If you are the Windows command interpreter, `Cmd.exe`, use the following guidelines to modify the example commands for your environment:

- Omit the “$” character at the beginning of the command. This is the Unix default prompt, equivalent to “>” in Windows.
- 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 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.

1.8 Controlling Input and Output Content Type

Input and output to the REST API comes in two forms: Document content and non-document data. Document content can be XML, JSON, text, or binary. Non-document data is anything that is not document content, such as document metadata, queries, query options, search results, and configuration data. Non-document data can usually be either XML or JSON, and you can choose which format you want to work with.

This section includes the following topics that explain how the REST API determines input and output content type, based on URI extension, HTTP headers, and the `format` request parameter.

- General Content Type Guidelines
- Details on Content Type Determination
- Example: Inserting and Reading a Document
- Example: Inserting and Reading Metadata
- Example: Documents With No or Unknown URI Extension
- Example: Mixing Document and Non-Document Data

1.8.1 General Content Type Guidelines

The following guidelines apply to specifying input and output content type for most requests:

- Document content: Rely on the MarkLogic Server MIME type mapping defined for the URI extension.
- Non-document data: Set the request Content-type and/or Accept headers. In most cases, this means setting the header(s) to `application/xml` or `application/json`.

The installation-wide MarkLogic Server MIME type mappings define associations between MIME type, URI extensions, and document format. For example, the default mappings associate the MIME type `application/pdf`, the "pdf" URI extension, and the binary document type. You can view, change, and extend the mappings in the "Mimetypes" section of the Admin Interface or using the XQuery functions `admin:mimetypes-get` and `admin:mimetypes-add`.

As long as your documents have URI extensions with a MIME type mapping and you set the HTTP Content-type and/or Accept headers consistent with your data, these guidelines are all you need for most requests. For situations that do not fit this model, see "Details on Content Type Determination" on page 26.

1.8.2 Details on Content Type Determination

This section provides a detailed description of how content type is determined. This information is useful for requests that do not conform to the guidelines in "General Content Type Guidelines" on page 25. For example, you might need this deeper understanding in the following situations:

- Reading or writing documents that have no URI extension or an unrecognized URI extension.
- Reading or writing document content and non-document data in the same request, such as reading a document and its metadata in a single request.
- Creating requests that have both input and output, such as a `POST:/v1/search` request that has a query in the POST body and search results in the response.
- Requesting non-document data through a browser. Browsers often do not give you full control over the HTTP headers.

The table below summarizes how input and output content type is determined, depending on type of data and the request context (input or output). The content type sources in the third column are listed from highest to lowest precedence. For example, for input document data, the URI extension mapping is used if possible; the Content-type header is only used if there is no mapping available.

Data Type	Context	Precedence of Content Type Sources
Document	Input	Primary: URI extension MIME type mapping. Fallback: Content-type header MIME type mapping. For multipart input, the request Content-type header must be `multipart/mixed`, so the Content-type header for each part specifies the MIME type of the content for that part.
	Output	URI extension MIME type mapping. Fallback: • For text, XML, and JSON documents, the document type (the type of root node on the document). • For binary documents, the Accept header MIME type mapping, except for requests with multipart output. • For multipart output, binary documents with no extension or an unknown extension: `application/x-unknown-content-type` by default.
Non-Document	Input	Primary: The `format` request parameter. Fallback: The Content-type header MIME type mapping. For multipart input, the request Content-type header must be `multipart/mixed, so` the Content-type header for each part specifies the MIME type of the content for that part.
	Output	Primary: The `format` request parameter. Fallback: The Accept header MIME type mapping, except for requests with multipart output.

The `format` request parameter is supported by most REST API methods that accept or produce non-document data. You can set it to one of a limited set of values, usually `xml` or `json`; see the API documentation for individual methods for allowed values.

Unless otherwise noted, the `format` parameter always overrides the Accept or Content-type header for non-document data. For requests that have both input and output, such as as `POST:/v1/search`, the `format` parameter value usually applies to both input and output non-document data. If you want to use different content types for input and output on such methods, use the HTTP headers only; do not set `format`.

Requests that accept or produce multipart data must set the Content-type header (multipart input) or Accept header (multipart output) to `multipart/mixed`. The `format` parameter enables you to specify a content type for non-document data in such cases. A multi-document write using `POST /v1/documents` is an example of an operation with multipart input. Reading a document and its metadata in a single request is an example of an operation with multipart output.

1.8.3 Example: Inserting and Reading a Document

This example demonstrates how the general content type guidelines apply to document content. The example relies on the pre-defined MIME type mapping between the `json` URI extension and the MIME type `application/json`.

The following command inserts a JSON document into the database with URI “example.json”. Because of the MIME type mapping, a JSON document is created, whether or not you specify `application/json` in the request Content-type header.

```
$ curl --anyauth --user user:password -X PUT -d '{"key":"value"}' \
  -i -H "Content-type: anything" \
  http://host:port/v1/documents?uri=example.json
```

The following command reads the document just inserted. Whether or not you set the Accept header to `application/json`, MarkLogic Server sets the response Content-type header to `application/json` because the URI extension is `json`.

```
$ curl --anyauth --user user:password -X GET -i \
  http://host:port/v1/documents?uri=example.json
...
HTTP/1.1 200 OK
vnd.marklogic.document-format: json
Content-type: application/json; charset=utf-8
...
{"key":"value"}
```

If the URI has no extension or there is no MIME type mapping defined for the extension, MarkLogic Server falls back on sources such as the HTTP Content-type header for input and the document type or Accept header for output. For details, see “Details on Content Type Determination” on page 26.

1.8.4 Example: Inserting and Reading Metadata

This example illustrates how the general content type guidelines apply to non-document data. The example inserts and reads document metadata.

The following command inserts metadata for a document. Assume the file `./mymatadata` contains a JSON representation of document metadata. The request Content-type header tells MarkLogic Server to interpret the metadata in the request body as JSON.

```
$ curl --anyauth --user user:password -X PUT -d @./mymetadata \
  -H "Content-type: application/json" \
  'http://host:port/v1/documents?uri=anything&category=metadata'
```

For a complete example, `PUT:/v1/documents` or see “Adding Metadata” on page 50.

The following command reads the metadata for a document. The Accept header tells MarkLogic Server to return the metadata as XML.

```
$ curl --anyauth --user user:password -X GET \
  -H "Accept: application/xml" \
  'http://host:port/v1/documents?uri=anything&category=metadata'
```

For a complete example, see `GET:/v1/documents` or “Retrieving Metadata About a Document” on page 59.

If you cannot control the Content-type header for input or the Accept header for output, you can use the `format` request parameter. For details, see “Details on Content Type Determination” on page 26.

```
http://host:port/v1/documents?uri=anything&category=metadata&format=json
```

1.8.5 Example: Documents With No or Unknown URI Extension

This example illustrates how the output content type is determined when reading a document with no URI extension or a URI extension that has no MIME type mapping.

The following command inserts a text document into the database at a URI that has no extension. Since there is no extension, MarkLogic Server uses the MIME type mapping defined for `text/plain` in the request Content-type header to determine the document type.

```
curl --anyauth --user user:password -X PUT -i \
  -d '{ "key" : "value" }' -H "Content-type: text/plain" \
  http://host:port/v1/documents?uri=no-extension
```

If you leave off the Content-type header or set it to value for which there is no MIME type mapping, a binary document is created because binary is the default document type when there is no extension or MIME type mapping.

The following command reads the document inserted above. The response Content-type header is `text/plain` because the root node of the document is a text node.

```
curl --anyauth --user user:password -X GET -i \
  http://host:port/v1/documents?uri=no-extension
```

For binary documents, the root document node type is too generic for most applications. You can use the Accept header to coerce the response Content-type header to a specific MIME type. The content is unaffected. For example, if you read a binary document with no extension that you know actually contains PDF, then the following command returns the document with a Content-type header of `application/pdf`.

```
curl --anyauth --user user:password -X GET -i \
  -H "Accept: application/pdf" \
  http://host:port/v1/documents?uri=no-extension
```

If you cannot control the Accept header, then the response Content-type for a binary document is `application/x-unknown-content-type` by default.

1.8.6 Example: Mixing Document and Non-Document Data

This example describes how content type is determined for requests that include both document and non-document data as input or as output. In this example, the non-document data is metadata for a document.

The following example command inserts an XML document and its metadata into the database. Assume the file "multipart-body" contains a `multipart/mixed` POST body with a part for the metadata and a part for the content. For a complete example, see "Loading Content and Metadata Using a Multipart Message" on page 53.

```
curl --anyauth --user user:password -X PUT -d @./multipart-body \
  -i -H "Content-type: multipart/mixed; boundary=BOUNDARY" \
  http://host:port/v1/documents?uri=example.xml
```

The metadata format is derived from the Content-type header on the metadata part in the POST body. The document content type is derived from the URI extension of `.xml`. If the document URI did not have an extension, the document content type would be derived from the Content-type header on the document part in the POST body.

The following example command reads a JSON document and its metadata. The response is multipart data, with one part containing the metadata and one part containing the document. Since the Accept header must be `multipart/mixed` in this case, the `format` parameter is used to request the metadata as JSON.

```
curl --anyauth --user user:password -X GET -i \
  -H "Accept: multipart/mixed; boundary=BOUNDARY" \
  'http://host:port/v1/documents?uri=example.json?format=json'
```

In the response, the Content-type header for the metadata part is set to `application/json` because of the `format` parameter value. The Content-type header for the document part is set to `application/json` because the document URI is `.json`. If the document URI had no extension, the Content-type header for the document part would still be `application/json` as long as the root node of the document indicates a JSON document.

For a complete example of reading a document and its metadata, see “Retrieving Content and Metadata in a Single Request” on page 59.

1.9 Working With JSON

The MarkLogic REST API support JSON in the following ways:

- Create, update, retrieve, delete, and query JSON documents.
- Create, update, retrieve, delete, and query document metadata using JSON.
- Specify queries, query options, and most configuration data in JSON.
- Request search results and most configuration data in JSON.
- Configure your REST API instance to return errors as JSON.

Note that the document manipulation operations do not include any implicit content conversions. For example, you cannot insert an XML document and request MarkLogic Server to convert it to a JSON document. Your application is responsible for inserting and retrieving content in a consistent way.

JSON documents are stored internally as XML. In most cases, this is transparent to your application. However, you might need to be aware of the internal representation if you create indexes on JSON keys, or create content transformations or resource service extensions. For more details, see:

- “Creating Indexes on JSON Keys” on page 150
- Working With JSON in the *Application Developer’s Guide*
- “Extending the REST API” on page 285
- “Working With Content Transformations” on page 269

1.10 Error Reporting

This section covers the error reporting conventions followed by the MarkLogic REST API.

If a request to a MarkLogic REST API instance fails, an error status code is returned and additional error detail is provided in the response body. You can configure your instance to return error details in either XML or JSON using the `error-format` configuration property. XML is the default content type. For details, see “Configuring Instance Properties” on page 41.

The following example shows error output in XML for a request specifying unsupported parameters. The return status code is 400 (Bad Request) and the details of the error, identifying the failure as a `REST-UNSUPPORTEDPARAM` exception, are contained in the response body.

```
HTTP/1.1 400 Bad Request
Content-type: application/xml
Server: MarkLogic
```

```
Content-Length: 333
Connection: close

<rapi:error xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:status-code>400</rapi:status-code>
  <rapi:status>Bad Request</rapi:status>
  <rapi:message-code>REST-UNSUPPORTEDPARAM</rapi:message-code>
  <rapi:message>REST-UNSUPPORTEDPARAM: (rest:UNSUPPORTEDPARAM)
Endpoint does not support query parameter: unknown</rapi:message>
</rapi:error>
```

The following example is the same error, with the error detail returned as JSON:

```
HTTP/1.1 400 Bad Request
Content-type: application/json
Server: MarkLogic
Content-Length: 206
Connection: close

{
  "error": {
    "status-code": "400",
    "status": "Bad Request",
    "message-code": "REST-UNSUPPORTEDPARAM",
    "message": "REST-UNSUPPORTEDPARAM: (rest:UNSUPPORTEDPARAM)
Endpoint does not support query parameter: unknown"
  }
}
```

Errors that can be corrected by the client application, such as an invalid parameter or an unsupported HTTP method, are usually reported as a 4XX error with a `REST-` or `RESTAPI-` message code.

Errors that cannot be addressed by the client application are usually reported as a 500 Internal Server Error. A 500 error does not necessarily mean that the problem cannot be corrected or that MarkLogic Server got an internal error. A 500 error usually indicates a problem that requires correction on the server host rather than in the client application.

An example of a 500 error that is correctable on the server side is failing to create an element range index required to support an operation. If a client application uses the `/search` service with a search constraint that requires a non-existent index, a 500 error is returned. To correct the error, an administrator would create the required index in MarkLogic Server.

XQuery and XSLT transforms and XQuery extensions you write should report errors using `RESTAPI-EXTNERR`, as described in “Reporting Errors” on page 274.

2.0 Administering MarkLogic REST API Instances

To use the MarkLogic REST API, you must create an *instance* of the API. The instance includes an HTTP App Server that exposes the API services, a URL rewriter, and the XQuery modules that implement the API.

- What is an Instance?
- Creating an Instance
- Removing an Instance
- Retrieving Configuration Information
- Configuring Instance Properties

2.1 What is an Instance?

Before you can use the MarkLogic REST API, you must create an *instance* that consists of an HTTP App Server specially configured to handle MarkLogic REST API requests, a content database, and a modules database.

> **Note:** Each REST API instance can host a single application. If you have multiple REST API applications, you must create an instance for each one, and each one must have its own modules database.

The content database can be created for you during the instance creation, or you can create it separately before making the instance. You can associate any content database with an instance. Administer your content database as usual, using the Admin Interface, XQuery Admin API, or REST Management API.

The modules database can be created for you during instance creation, or you can create it separately before making the instance. If you choose to pre-create the modules database, it must not be shared across instances. Special code is inserted into the modules database during instance creation. The modules database also holds any persistent query options, extensions, and content transformations you create for your application.

You cannot pre-configure the App Server. However, once it is created, you can further customize properties such as request timeouts using the Admin Interface or XQuery Admin API.

2.2 Creating an Instance

You can create a REST API instance either by making an HTTP request or interactively, as described in the following topics:

- Creating an Instance Interactively
- Creating an Instance Using HTTP
- Example: Creating an Instance Using HTTP

Note: Each REST API instance can host a single application. If you have multiple REST API applications, you must create an instance for each one, and each one must have its own modules database.

2.2.1 Creating an Instance Interactively

You can create an instance interactively from the MarkLogic Server Welcome page. A complete walkthrough of this method is available in "Getting Started with the MarkLogic REST API" on page 12.

To create an instance with this method, use your browser to navigate to the Welcome page on port 8000 and authenticate as a user with the `rest-reader` role. For example, navigate to the following location:

```
http://your-marklogic-host:8000
```

On the Welcome page, click the link to Create a REST API Instance in the Integrate section. Fill in the information about your instance, such as name, port, and content database, and click the Create REST API Instance.

If you require finer control, such as specifying a different group, you must use a POST request to the `/rest-apis` service to create your instance. For details, see "Creating an Instance Using HTTP" on page 34.

You can also create an instance interactively as part of the Information Studio flow. For details, see Creating a REST API Instance in the *Information Studio Developer's Guide*.

2.2.2 Creating an Instance Using HTTP

This section focuses on using the `/rest-apis` service directly to create an instance. This method of creating an instance is easily scriptable.

To create a new instance using HTTP, send a POST request to the `/rest-apis` service on port 8002 with a URL of the form:

```
http://host:8002/version/rest-apis
```

For the current *version*, see "Specifying the REST API Version" on page 23.

The POST body should contain instance configuration information in XML or JSON. Set the HTTP `Content-type` header to `application/xml` or `application/json` to indicate the content format. MarkLogic Server responds with HTTP status 201 if the instance is successfully created.

The configuration data must specify a name for the instance and can also specify a group name, database name, modules database name, and port.

The following example configuration creates an instance named RESTstop on port 8020, attached to the Documents database, using the default modules database, RESTstop-modules.

XML	JSON

<rest-api xmlns="http://marklogic.com/rest-api">
  <name>RESTstop</name>
  <database>Documents</database>
  <port>8020</port>
</rest-api>
	{ "rest-api": {
	  "name": "RESTstop",
	  "database": "Documents",
	  "port": "8020"
	} }

The following example creates an instance using the above XML configuration information, saved to a file named “config.xml”. To use the JSON example configuration data, change the Content-type header value to `application/json`.

```
$ cat config.xml
<rest-api xmlns="http://marklogic.com/rest-api">
  <name>RESTstop</name>
  <database>Documents</database>
  <port>8020</port>
</rest-api>

# Windows users, see Modifying the Example Commands for Windows
$ curl -X POST --anyauth --user user:password -d @"./config.xml" \
    -H "Content-type: application/xml" \
    http://localhost:8002/v1/rest-apis
```

Once the instance is created, you can customize and administer the App Server portion using the Admin UI, just as you can with any MarkLogic Server App Server.

The following table provides more details on the configuration components.

Element/key name	Default	Description
`name`	none	Required. The name of the instance.
`group`	Default	Optional. The group in which to create the App Server.

Element/key name	Default	Description
`database`	*instance_name*	Optional. The name of the content database to associate with the instance. If the database does not exist, MarkLogic Server creates a new database with default settings and one forest.
`modules-database`	*instance_name*-modules	Optional. The name of the modules database to associate with the instance. If the database does not exist, MarkLogic Server creates a new database with default settings and one forest.
`port`	The next available port number, starting with 8003.	Optional. The port number to associate with the App Server.

2.2.3 Example: Creating an Instance Using HTTP

Follow this procedure to create a "rest-example" database with one forest and a "rest-example" HTTP App Server attached to the database. The example uses a minimal JSON configuration as input. The equivalent XML configuration is shown after the procedure.

1. Create the instance and database by sending a POST request to `/rest-apis` on port 8002. MarkLogic Server responds with status code 201 (Created). For example:

```
$ curl --anyauth --user user:password -i -X POST \
    -d'{"rest-api":{"name":"rest-example"}}' \
    -H "Content-type: application/json" \
    http://localhost:8002/v1/rest-apis
```

   By specifying only the instance name, we request MarkLogic Server to create a database with the same name, including one forest, and to use the next available port above 8002.

2. Navigate to the Admin Interface in your browser:

```
http://localhost:8001
```

   Observe the creation of a database named "rest-example" with a forest named "rest-example-1", and an HTTP App Server called "rest-example" attached to this database. A modules database is also created for the instance.

3. Test the instance by navigating to the following URL in your browser. (Your port number may differ if port 8003 was already in use). When prompted for a username and password, authenticate with a user that has the `rest-reader` role.

```
http://localhost:8003
```

If the instance is properly configured, you will see a short HTML summary of capabilities provided by the instance.

If you prefer to configure your instance using XML, use the following for the POST body and change the HTTP Content-type header to `application/xml` in Step 1, above.

```
<rest-api xmlns="http://marklogic.com/rest-api">
  <name>rest-example</name>
</rest-api>
```

For an example of creating an instance interactively, see “Create a REST API Instance” on page 13.

2.3 Removing an Instance

To remove an instance of the MarkLogic REST API, send a DELETE request to the /rest-apis service on port 8002. To remove the instance but leave the content database intact, send the request with a URL of the form:

```
http://host:8002/version/rest-apis/instance-name
```

`instance-name` is the name supplied when the instance is created, which is also the name of the App Server associated with the instance. MarkLogic Server responds with 204 on success and 400 (“Bad Request”) if the instance is not found or more than one occurrence of the instance name is found.

Note: Deleting an instance causes MarkLogic Server to restart.

To also remove the content and/or modules database and forests associated with the instance, use the `include` request parameter.

Note: All content in the removed databases is lost.

The following example removes the “RESTstop” instance, including the content and modules databases, and the forests attached to the databases:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl -X DELETE --anyauth --user user:password \
  'http://localhost:8002/v1/rest-apis/RESTstop?include=content&include
=modules'
```

If the instance name is not unique across groups, use the `group` request parameter to differentiate the target instance. The following URL requests removal of the instance of “RESTstop” in the “stage” group:

```
$ curl -X DELETE --anyauth --user user:password \
  http://localhost:8002/v1/rest-apis/RESTstop?group=stage
```

2.4 Retrieving Configuration Information

You can retrieve instance configuration information, as XML or JSON, by sending a GET request to the /rest-apis service on port 8002.

- Retrieving Configuration for All Instances
- Retrieving Instance Configuration by Content Database
- Retrieving Instance Configuration by Instance Name

2.4.1 Retrieving Configuration for All Instances

To retrieve configuration details about an instance, send a GET request to the `/rest-apis` service on port 8002 with a URL of the form:

```
http://host:8002/version/rest-apis
```

MarkLogic Server responds with configuration information about all instances, in XML or JSON. Use the Accept header or format request parameter to select the output content type. For details, see “Controlling Input and Output Content Type” on page 25.

The following example requests information in JSON about all instances.

```
# Windows users, see Modifying the Example Commands for Windows
$ curl -X GET --anyauth --user user:password \
    -H "Accept: application/json" \
    http://localhost:8002/v1/rest-apis
```

The response includes configuration details about 2 instances, named RESTstop and OscarsREST.

```
{
  "rest-apis":[
    {
      "name": "RESTstop",
      "group": "Default",
      "database": "bill",
      "modules-database": "RESTstop-modules",
      "port": "8003"
    },
    {
      "name":"OscarsREST",
      "group": "Default",
      "database":"Oscars",
      "modules-database": "OscarsREST-modules",
      "port":"8006"
    }
  ]
}
```

Requesting the same data as XML results in the following:

```
<rapi:rest-apis xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:rest-api>
    <rapi:name>RESTstop</rapi:name>
    <rapi:group>Default</rapi:group>
    <rapi:database>bill</rapi:database>
    <rapi:modules-database>RESTstop-modules</rapi:modules-database>
    <rapi:port>8003</rapi:port>
  </rapi:rest-api>
  <rapi:rest-api>
    <rapi:name>OscarsREST</rapi:name>
    <rapi:group>Default</rapi:group>
    <rapi:database>Oscars</rapi:database>
    <rapi:modules-database>OscaresREST-modules</rapi:modules-database>
    <rapi:port>8006</rapi:port>
  </rapi:rest-api>
</rapi:rest-apis>
```

You can also constrain the results to a particular content database or a specific instance. See “Retrieving Instance Configuration by Content Database” on page 39 and “Retrieving Instance Configuration by Instance Name” on page 40.

2.4.2 Retrieving Instance Configuration by Content Database

To retrieve configuration details instances associated with a specific database, send a GET request to the /rest-apis service on port 8002 with a URL of the form:

```
http://host:8002/version/rest-apis?database=database_name
```

MarkLogic Server responds with configuration information about all instances serving the named database, in XML or JSON. Use the Accept header or format request parameter to select the output content type. For details, see “Controlling Input and Output Content Type” on page 25.

The following example requests information in JSON about instances associated with the database named “bill”:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl -X GET --anyauth --user user:password \
    -H "Accept: application/json" \
    'http://localhost:8002/v1/rest-apis?database=bill'
```

The response contains data similar to the following, depending on the requested response format.

Format	Response Data

XML	`<rapi:rest-apis xmlns:rapi="http://marklogic.com/rest-api">`
	`  <rapi:rest-api>`
	`  <rapi:name>RESTstop</rapi:name>`
	`  <rapi:group>Default</rapi:group>`
	`  <rapi:database>bill</rapi:database>`
	`  <rapi:modules-database>`
	`  RESTstop-modules`
	`  </rapi:modules-database>`
	`  <rapi:port>8003</rapi:port>`
	`  </rapi:rest-api>`
	`</rapi:rest-apis>`

JSON	`{ "rest-apis": [`
	`  {`
	`  "name": "RESTstop",`
	`  "group": "Default",`
	`  "database": "bill",`
	`  "modules-database": "RESTstop-modules",`
	`  "port": "8003"`
	`  }`
	`] }`

2.4.3 Retrieving Instance Configuration by Instance Name

To retrieve configuration details about a specific instance, send a GET request to the `/rest-apis` service on port 8002 with a URL of the form:

```
http://host:8002/version/rest-apis/instance_name
```

MarkLogic Server responds with configuration information about the named instance, in XML or JSON. Use the Accept header or format request parameter to select the output content type. For details, see “Controlling Input and Output Content Type” on page 25.

The following example requests information in JSON about the “RESTstop” instance:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl -X GET --anyauth --user user:password \
    -H "Accept: application/json" \
    http://localhost:8002/v1/rest-apis/RESTstop
```

The response contains data similar to the following, depending on the requested response format.

Format	Response Data

XML	`<rapi:rest-api xmlns:rapi="http://marklogic.com/rest-api">`
	`  <rapi:name>RESTstop</rapi:name>`
	`  <rapi:group>Default</rapi:group>`
	`  <rapi:database>bill</rapi:database>`
	`  <rapi:modules-database>RESTstop-modules</rapi:modules-database>`
	`  <rapi:port>8003</rapi:port>`
	`</rapi:rest-api>`

JSON	`{`
	`  "name": "RESTstop",`
	`  "group": "Default",`
	`  "database": "bill",`
	`  "modules-database": "RESTstop-modules",`
	`  "port": "8003"`
	`}`

Use the `group` parameter to disambiguate the instance if there is more than one instance or App Server with the same *instance_name* in your MarkLogic Server cluster.

2.5 Configuring Instance Properties

You can use the `/config/properties` and `/config/properties/{name}` services to manage properties that control the behavior of a MarkLogic REST API instance. The following topics are covered:

- Instance Configuration Properties
- Listing Instance Property Settings
- Setting Instance Properties
- Resetting Instance Properties

2.5.1 Instance Configuration Properties

You can read and set the following instance properties to control global behaviors of your MarkLogic REST API instance:

Name	Description
`content-versions` (deprecated)	Deprecated as of MarkLogic Server v7.0-3. Use the `update-policy` configuration property instead.
`debug`	Turn debug logging on and off. Allowed values: `true`, `false`. Default: `false`.
`document-transform-out`	The name of a default content transformation to apply when retrieving documents from the database using `/documents`. If set, this must be the name of a transform installed using `/config/transforms/{name}`. The default transform is applied before any transform specified with the `transform` request parameter. Default: apply no transformations. For details, see "Working With Content Transformations" on page 269.
`error-format`	The content type of errors returned by the MarkLogic REST API. Allowed values: `xml`, `json`. Default: `xml`.
`update-policy`	This property controls the availability and behavior of conditional GET, PUT, POST, and DELETE on the `/documents` service and metadata merging policy. Allowed values: `version-required`, `version-optional`, `merge-metadata`, `overwrite-metadata`. Default: `merge-metadata`. For details, see "Using Optimistic Locking to Update Documents" on page 114 and "Client-Side Cache Management Using Content Versioning" on page 120.
`validate-options`	Whether or not to validate query options when they are created or updated using `/config/query*`. When option validation is enabled, improperly structured query options are rejected. Allowed values: `true`, `false`. Default: `true`.

2.5.2 Listing Instance Property Settings

To list all available properties and their settings, send a GET request of the following form to the `/config/properties` service:

```
http://host:port/version/config/properties
```

To retrieve the setting of a single property, send a GET request of the following form, where *property-name* is one of the property names listed in “Instance Configuration Properties” on page 42.

```
http://host:port/version/config/properties/property-name
```

You can retrieve property settings as XML or JSON. Use the Accept header or `format` request parameter to select the output content type. For details, see “Controlling Input and Output Content Type” on page 25.

The following example command retrieves all property settings as XML:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password \
    http://localhost:8003/v1/config/properties

<rapi:properties xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:content-versions>none</rapi:content-versions>
  <rapi:debug>false</rapi:debug>
  <rapi:document-transform-out/>
  <rapi:error-format>xml</rapi:error-format>
  <rapi:validate-options>true</rapi:validate-options>
</rapi:properties>
```

The following example retrieves the same information as JSON:

```
$ curl --anyauth --user user:password \
    -H "Accept: application/json" \
    http://localhost:8003/v1/config/properties

{
  "content-versions": "none",
  "validate-options": true,
  "document-transform-out": "",
  "debug": false,
  "error-format": "xml"
}
```

The following example retrieves the setting for just the “debug” property as JSON:

```
$ curl --anyauth --user user:password \
    -H "Accept: application/json" \
    http://localhost:8003/v1/config/properties/debug

{ "debug" : false }
```

The equivalent XML output is shown below:

```
<rapi:debug xmlns:rapi="http://marklogic.com/rest-api">
  false
</rapi:debug>
```

2.5.3 Setting Instance Properties

You can update instance property settings by sending a PUT request to either `/config/properties` or `/config/properties/{name}`. Using `/config/properties` allows you to set multiple properties in a single call.

To set one or more properties, send a PUT request to `/config/properties` with a URL of the form:

```
http://host:port/version/config/properties
```

The body of the request should contain an XML or JSON properties structure containing only the properties you wish to change. When using XML, the `<properties/>` structure must be in the namespace "http://marklogic.com/rest-api". See the examples below.

To set a specific property, send a PUT request of the following form to `/config/properties/{name}`, where property-name is the property you wish to change. For a list of property names, see "Instance Configuration Properties" on page 42.

```
http://host:port/version/config/properties/property-name
```

Property settings can be specified in either XML or JSON. Use the Content-type header or `format` request parameter to select the output content type. For details, see "Controlling Input and Output Content Type" on page 25..

The following example sets the "debug" and "validate-options" properties using XML input to `/config/properties`:

```
$ cat props.xml
<properties xmlns="http://marklogic.com/rest-api">
  <debug>true</debug>
  <validate-options>false</validate-options>
</properties>

# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user admin:adm1n -X PUT -d@"./props.xml" \
    -H "Content-type: application/xml" \
    http://localhost:8003/v1/config/properties
```

The equivalent JSON input is shown below:

```
{
  "debug" : "true",
  "validate-options" : "true"
}
```

The following example sets the "debug" property to true using XML input to `/config/properties/{name}`:

```
$ cat debug-prop.xml
<debug xmlns="http://marklogic.com/rest-api">
  true
</debug>

$ curl --anyauth --user admin:adm1n -X PUT -d@"./debug-prop.xml" \
    -H "Content-type: application/xml" \
    http://localhost:8003/v1/config/properties/debug
```

The equivalent JSON input is:

```
{"debug" : "true"}
```

2.5.4 Resetting Instance Properties

To reset all properties to their default values, send a DELETE request of the following form to /config/properties:

```
http://host:port/version/config/properties
```

The following example command resets all instance properties:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X DELETE \
    http://localhost:8003/v1/config/properties
```

To reset just one property to its default value, send a DELETE request of the following form to /config/properties/{name}:

```
http://host:port/version/config/properties/property-name
```

Where property-name is one of the properties listed in "Instance Configuration Properties" on page 42.

The following example command resets just the "debug" property:

```
$ curl --anyauth --user user:password -X DELETE \
    http://localhost:8003/v1/config/properties/debug
```

3.0 Manipulating Documents

This chapter discusses the following topics related to using the MarkLogic REST API to create, read, update and delete documents and metadata:

- Summary of Document Management Services
- Loading Content into the Database
- Retrieving Documents from the Database
- Partially Updating Document Content or Metadata
- Performing a Lightweight Document Check
- Removing Documents from the Database
- Using Optimistic Locking to Update Documents
- Client-Side Cache Management Using Content Versioning
- Working with Binary Documents
- Working with Metadata

3.1 Summary of Document Management Services

This section gives a brief summary of the MarkLogic REST API services available for creating, reading, updating, and delete documents. This section covers the following topics:

- Summary of the /documents Service
- Summary of the /graphs Service

3.1.1 Summary of the /documents Service

Use the `/documents` service to create, read, update, and delete document content and metadata for XML, JSON, text, and binary documents, including XML documents containing embedded semantic RDF triple data. To load documents containing only RDF triples, use the `/graph` service; for details, see "Summary of the /graphs Service" on page 48.

The following table summarizes the supported operations:

Operation	Method	Description
Create/Update	PUT	Create or update the content or metadata of a document.
Create	POST	Create a document with a URI automatically generated by MarkLogic Server.
Partial Update	POST	Update a portion of the content of an XML or JSON document, or update metadata of any document.
Multi-Document Write	POST	Create or update multiple the content and/or metadata of multiple documents in a single request.
Retrieve	GET	Retrieve content and/or metadata for one or more documents.
Delete	DELETE	Remove a document, or remove or reset document metadata.
Test	HEAD	Test for the existence of a document or determine the size.

The service supports the following additional document features through request parameters:

- Transaction control
- Transformation of content during ingestion
- Content repair during ingestion
- Transformation of results during document insertion or retrieval
- Conditional document insertion using optimistic locking
- Conditional reads based on content versioning

Note: XML, JSON and text documents must use UTF-8 encoding.

Note: The service does not support content conversions except through custom transformations. For example, you cannot retrieve an XML document as JSON.

This chapter only covers single-document operations. For multi-document operations, see "Reading and Writing Multiple Documents" on page 206.

3.1.2 Summary of the /graphs Service

Use the `/graphs` service to create, read, update, and delete documents containing RDF triples. For triples embedded in XML documents, use the `/documents` service.

The following table summarizes the supported operations:

Operation	Method	Description
Create/Update	PUT	Create or replace triples in a named graph or the default graph.
Update	POST	Merge triples into a named graph or the default graph.
Retrieve	GET	Retrieve a named graph or the default graph.
Delete	DELETE	Remove triples in a graph.
Test	HEAD	Test for the existence of a graph in the database or retrieve the headers that would be returned by a GET request.

Though a graph can include both triples embedded in XML documents and triples from a pure triples document (one with a `<sem:triples>` root element), the `/graphs` service does not operate on embedded triples. For example, if you make a DELETE request on a graph that includes embedded triples, the embedded triples and their containing document are unaffected, and therefore, the graph continues to exist.

For details, see *Semantics Developer's Guide*.

3.2 Loading Content into the Database

This section focuses on ingesting whole documents and their metadata into the database. To learn about modifying just a portion of a document, see "Partially Updating Document Content or Metadata" on page 61.

To insert content or metadata into the database, make a PUT or POST request to the `/documents` service. This section covers the following topics:

- Loading Content
- Adding Metadata
- Loading Content and Adding Metadata in the Same Request
- Automatically Generating Document URIs

- Loading JSON Documents Into the Database
- Loading Triples
- Controlling Access to Documents Created with the REST API
- Transforming Content During Ingestion

3.2.1 Loading Content

This section describes how to insert or update an entire document at a user-defined database URI. If you want to have MarkLogic Server generate document URIs automatically, see "Automatically Generating Document URIs" on page 55. You can also modify only a portion of a document; for details, see "Partially Updating Document Content or Metadata" on page 61. For loading multiple documents in a single request, see "Reading and Writing Multiple Documents" on page 206.

To insert or update an XML, JSON, text, or binary document, make a PUT request to a URL of the form:

```
http://host:port/version/documents?uri=document_uri
```

When constructing the request:

1. Set the `uri` parameter to the URI of the destination document in the database.

2. Place the content in the request body.

3. Specify the MIME type of the content in the `Content-type` HTTP header.

   Note: XML, JSON and text documents must be use UTF-8 encoding.

Documents you create with the MarkLogic REST API have a read permission for the `rest-reader` role and an update permission for the `rest-writer` role. To restrict access, use custom roles. For details, see "Controlling Access to Documents Created with the REST API" on page 57.

The following example command sends a request to insert the contents of the file `./my.xml` into the database as an XML document with URI `/xml/example.xml`:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT -T ./my.xml \
    -H "Content-type: application/xml" \
    http://localhost:8003/v1/documents?uri=/xml/example.xml
```

If the MIME type is not set in the HTTP Content-type header, MarkLogic Server uses the file extension on the document URI to determine the content format, based on the MIME type mappings defined for your installation.

You can also set metadata such as collections, permissions, and named properties when loading content. See “Loading Content and Adding Metadata in the Same Request” on page 52.

You can have MarkLogic Server generate document URIs for you. For details, see “Automatically Generating Document URIs” on page 55.

For additional information about working with JSON documents, see “Loading JSON Documents Into the Database” on page 56.

3.2.2 Adding Metadata

This section describes inserting or updating metadata using a PUT request to `/documents`. The following topics are covered:

- Inserting or Updating Metadata
- Example: Replacing One Metadata Category Using XML
- Example: Replacing Multiple Metadata Categories Using XML
- Example: Replacing Multiple Metadata Categories Using JSON

3.2.2.1 Inserting or Updating Metadata

To insert or update only metadata for a document, make a PUT request to a URL of the form:

```
http://host:port/version/documents?uri=document_uri&category=metadata_
category
```

Where `category` can appear multiple times, with the values described in “Metadata Categories” on page 124.

> **Note:** You cannot supply metadata via request parameters when there is no document content. In this case, you must place the XML or JSON metadata in the request body.

When constructing the request:

1. Set the `category` parameter to the type of metadata to insert or replace. Specify `category` multiple times to include more than one type of metadata.

2. Place the metadata in the request body. For format details, see “Working with Metadata” on page 124.

3. Specify the metadata format in the HTTP `Content-type` header or the `format` parameter. You may supply either JSON or XML metadata.

Metadata category merging is not available. A PUT request for metadata completely replaces each category of metadata specified in the request. For example, a PUT request for collections replaces all existing collections. When the category is `metadata`, all metadata is replaced or reset to default values.

Note: When setting permissions, at least one update permission must be included.

Metadata can be supplied as either XML or JSON. Use the `format` parameter or the `Content-type` header to specify the format of the metadata. If both `format` and `Content-type` are given, `format` takes precedence. If neither `format` nor `Content-type` is specified, XML is assumed. For details on formats, see "Working with Metadata" on page 124.

Metadata for categories other than those named by the `category` parameter(s) are ignored. For example, if the request body contains metadata for both collections and properties, but only `category=collections` is given in the URL, then only the collections are updated.

3.2.2.2 Example: Replacing One Metadata Category Using XML

The following example places the document with URI /xml/example.xml into the "interesting" collection by specifying `category=collections`. The document is removed from any other collections. The metadata XML in the request body defines the name of the collection(s).

```
$ cat metadata.xml
<rapi:metadata xmlns:rapi="http://marklogic.com/rest-api"
               xmlns:prop="http://marklogic.com/xdmp/property">
  <rapi:collections>
    <rapi:collection>interesting</rapi:collection>
  </rapi:collections>
</rapi:metadata>

# Windows users, see Modifying the Example Commands for Windows
$ curl -X PUT -T ./metadata.xml -H "Content-type: application/xml" \
    --anyauth --user user:password \
    'http://localhost:8003/v1/documents?uri=/xml/example.xml&category=
collections'
```

3.2.2.3 Example: Replacing Multiple Metadata Categories Using XML

This example replaces multiple types of metadata on the document with URI /xml/example.xml by specifying multiple category parameters. The metadata in the request body defines a collection name ("interesting") and a property ("my-property"). The request URL includes `category=collections` and `category=properties`. Any collections or properties previously set for the document /xml/example.xml are replaced with the new values.

```
$ cat > metadata.xml
<rapi:metadata xmlns:rapi="http://marklogic.com/rest-api"
               xmlns:prop="http://marklogic.com/xdmp/property">
  <rapi:collections>
    <rapi:collection>interesting</rapi:collection>
  </rapi:collections>
```

```
  <prop:properties>
    <my-property>value</my-property>
  </prop:properties>
</rapi:metadata>

# Windows users, see Modifying the Example Commands for Windows
$ curl -X PUT -T ./metadata.xml -H "Content-type: application/xml" \
    --anyauth --user user:password \
    'http://localhost:8003/v1/documents?uri=/xml/example.xml&category=
collections&category=properties'
```

3.2.2.4 Example: Replacing Multiple Metadata Categories Using JSON

This example replaces multiple types of metadata on the document with URI /xml/example.xml by specifying multiple category parameters. The JSON metadata in the request body defines a collection name ("interesting") and a property ("my-property"). The request URL includes `category=collections` and `category=properties`. Any collections or properties previously set for the document /xml/example.xml are replaced with the new values.

```
$ cat > metadata.json
{
  "collections":["interesting"],
  "properties": {
    "my-property":"name"
  }
}

# Windows users, see Modifying the Example Commands for Windows
$ curl -X PUT -T ./metadata.json --anyauth --user user:password \
    -H "Content-type: application/json" \
    'http://localhost:8003/v1/documents?uri=/xml/example.xml&category=
collections&category=properties'
```

The example uses the Content-type header to communicate the metadata content-type to MarkLogic Server. The format can also be specified through the `format` request parameter.

3.2.3 Loading Content and Adding Metadata in the Same Request

You can update content and metadata for a document in a single request using two methods:

- Loading Content and Metadata Using Request Parameters
- Loading Content and Metadata Using a Multipart Message

For loading multiple content and/or metadata for multiple documents in a single request, see "Reading and Writing Multiple Documents" on page 206.

3.2.3.1 Loading Content and Metadata Using Request Parameters

Use this method when you want to specify metadata using request parameters. To load content and include metadata in the request parameters, send a PUT request of the following form to the/documents service:

```
http://host:port/version/documents?uri=doc_uri&metadata_param=value
```

Where *metadata_param* is one of `collection`, `perm:role`, `prop:name`, `quality`. For example, to set the property named "color" to red, include `prop:color=red` in the URL.

When constructing the request:

1. Set the `uri` parameter to the URI of the destination document in the database.
2. Place the content in the request body.
3. Specify the MIME type of the content in the `Content-type` HTTP header.
4. Specify the value of one or more metadata categories through request parameters, such as `collection` or `prop`.

If the MIME type is not set in the Content-type header, MarkLogic Server uses the file extension on the document URI to determine the content format, based on the MIME type mapping defined for the database. MIME type mappings for file suffixes are defined in the Admin Interface.

The following example inserts a binary document with the URI `/images/critter.jpg` into the database, adds it to the "animals" collection, and sets a "species" property:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl -X PUT -T ./critter.jpg --anyauth --user user:password \
    -H "Content-type: image/jpeg" \
    'http://localhost:8003/v1/documents?uri=/images/critter.jpg&collec
tion=animals&prop:species="canus lupus"'
```

You can also create a block of metadata as JSON or XML and pass it in the request body with the content. See "Loading Content and Metadata Using a Multipart Message" on page 53.

3.2.3.2 Loading Content and Metadata Using a Multipart Message

Use this method when you want to insert or update both content and metadata for a document in a single request, and you want to specify the metadata as JSON or XML in the request body. You can also specify metadata using request parameters. For details, see "Loading Content and Metadata Using Request Parameters" on page 52.

Construct a PUT request with a `multipart/mixed` message body where the metadata is in the first part and the document content is in the second part of the request body. The request URL is of the form:

```
http://host:port/version/documents?uri=doc_uri&category=content&category=
metadata_category
```

Where `category` can appear multiple times, with the values described in "Metadata Categories" on page 124.

When constructing the request:

1. Set the `uri` parameter to the URI of the destination document in the database.
2. Set `category=content` in the request URL to indicate content is included in the body.
3. Set additional `category` parameters to indicate the type(s) of metadata to add or update.
4. Specify `multipart/mixed` in the HTTP Content-type header for the request.
5. Set the part boundary string in the HTTP Content-type header to a string of your choosing.
6. Set the `Content-type` of the first part to either `application/xml` or `application/json` and place the XML or JSON metadata in the part body.
7. Set the `Content-type` of the second part to the MIME type of the content and place the content in the part body.

For details on metadata formats, see "Working with Metadata" on page 124.

Note: Metadata must always be the first part of the multipart body.

The following example inserts an XML document with the URI `/xml/box.xml` into the database and adds it to the "shapes" and "squares" collection. The collection metadata is provided in XML format in the first part of the body, and the content is provided as XML in the second part.

```
$ cat ./the-body
--BOUNDARY
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<metadata xmlns="http://marklogic.com/rest-api">
  <collections>
    <collection>shapes</collection>
    <collection>squares</collection>
  </collections>
</metadata>
--BOUNDARY
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<data>
  <mybox>
    This is my box. There are many like it, but this one is mine.
  </mybox>
</data>
--BOUNDARY--
```

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT \
    --data-binary @./the-body \
    -H "Content-type: multipart/mixed; boundary=BOUNDARY" \
    'http://localhost:8003/v1/documents?uri=/xml/box.xml&category=coll
ections&category=content'
```

You can also pass metadata in URL parameters instead of in the request body. See “Loading Content and Metadata Using Request Parameters” on page 52.

3.2.4 Automatically Generating Document URIs

MarkLogic Server can automatically generate databse URIs for documents inserted with the REST API. You can only use this feature to create new documents. To update existing documents, you must know the URI; for details, see “Loading Content” on page 49.

To use this feature, send a POST request to `/documents` that includes the `extension` request parameter and does not include the `uri` request parameter. That is, a POST request with a URL of the following form:

```
http://host:port/version/documents?extension=file-extension
```

Where extension specifies the document URI file prefix, such as `xml` or `txt`. Optionally, you can also include a database directory prefix with the `directory` request parameter.

Note: The directory prefix should end in a forward slash (`/`).

MarkLogic Server returns the auto-generated URI in the Location header of the response.

The following example inserts a document into the database with a URI of the form `/my/directory/auto-generated.xml`:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X POST -d@'./my-content' -i \
  -H "Content-type: application/xml" \
  'http://localhost:8003/v1/documents?extension=xml&directory=/my/dire
ctory/'

HTTP/1.1 201 Document Created
Location: /my/directory/6041706572796142832.xml
...
```

You can use the same features with generated URIs as when loading documents with user-defined URIs. For example, you can insert both documents and metadata and apply content transformations. For details, see “Loading Content into the Database” on page 48 and the *MarkLogic REST API Reference*.

3.2.5 Loading JSON Documents Into the Database

You can insert JSON documents into the database using `/documents`. To insert JSON content, simply specify a Content-type of `application/json` when inserting the document. For example:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT -T ./my.json \
    -H "Content-type: application/json" \
    http://localhost:8003/v1/documents?uri=/json/example.json
```

Internally, MarkLogic Server converts your JSON document into an XML representation so you can use the powerful query capabilities of MarkLogic Server on your JSON content. Usually, you do not need to be aware of this representation, though you can examine it using Query Console's Explore feature. You might need to examine the internal representation if you need to define a range index on JSON content. For details, see "Creating Indexes on JSON Keys" on page 150.

Note: Element names in the XML representation of a JSON document are in the namespace "http://marklogic.com/xdmp/json/basic". You must use this namespace when defining indexes on keys in JSON documents.

If you use content transformations, the transformations are applied to the XML representation. For details, see "Working With Content Transformations" on page 269.

For more information on JSON support in MarkLogic Server, see Working With JSON in the *Application Developer's Guide*.

3.2.6 Loading Triples

You can use the `/graphs` service to load semantic triples into the database in several formats. The service implements the W3C Graph Store Protocol, described by the following specification:

http://www.w3.org/TR/2013/REC-sparql11-http-rdf-update-20130321/

To load triples, send a PUT or POST request to the /graphs service with one of the following forms of URL, depending upon whether you wish to address a specific named graph or the default graph:

```
http://host:port/version/graphs?graph=graph-uri

http://host:port/version/documents?default
```

Note: You must enable the collection lexicon on your database before you can use the semantics REST services or use the GRAPH '?g' construct in a SPARQL query.

For details, see Loading Triples Using the REST API and Supported RDF Triple Serializations in *Semantics Developer's Guide*.

3.2.7 Controlling Access to Documents Created with the REST API

By default, documents you create with the MarkLogic REST API have a read permission for the `rest-reader` role and an update permission for the `rest-writer` role. A user with the `rest-reader` role can read all documents created with the REST API, and a user with the `rest-writer` role can update all documents created with the REST API.

To enable users to create and update documents using the REST API yet restrict access, use custom roles with the `rest-reader` and `rest-writer` execute privileges and suitable default permissions, rather than relying on the pre-defined `rest-reader` and `rest-writer` roles. The `rest-reader` and `rest-writer` privileges grant users permission to execute REST API code for reading and writing documents, while the default permission controls access to a document whether it is through the REST API or through other code running on MarkLogic Server. For details, see the *Understanding and Using Security Guide*.

For example, suppose you have two groups of users, A and B. Both can create documents using the REST API, but Group A users should not be able to read documents created by Group B, and vice versa. You can implement these restrictions in the following way:

1. Create a GroupA security role.

2. Assign the `rest-reader` and `rest-writer` execute privileges to the GroupA role. Use the privileges, not the base roles. That is, assign these privileges to the role:

   ```
   http://marklogic.com/xdmp/privileges/rest-reader
   http://marklogic.com/xdmp/privileges/rest-writer
   ```

3. If you also want to enable the execution of REST resource extensions, assign the `rest-extension-user` role to the GroupA role. (The `rest-extension-user` role provides a base role, not a privilege.)

4. Give the GroupA role suitable default permissions. For example, set the default permissions of the role to `update` and `read`.

5. Assign the GroupA role to the appropriate users.

6. Repeat Steps 1-4 for a new GroupB role and assign GroupB to the appropriate users.

Now, users with the GroupA role can create documents with the REST API and read or update them, but users with the GroupB role have no access to documents created by GroupA. Similarly, users with the GroupB role can create documents and read or update them, but users with the GroupA role have no access to documents created by GroupB users. A user with the default `rest-reader` role, however, can read documents created by both GroupA and GroupB users.

Other security configurations are possible. For more details, see the *Understanding and Using Security Guide*.

3.2.8 Transforming Content During Ingestion

You can transform content during ingestion by applying custom transform. A transform is an XQuery module or XSLT stylesheet you write and install using `/config/transforms/{name}`. For details, see "Working With Content Transformations" on page 269.

To apply a transform when creating or updating a document, add the `transform` parameter to your request. If the transform expects parameters, specify them using `trans:`*`paramName`* parameters. That is, your request should be of the form:

```
http://host:port/version/v1/documents?...&transform=name&trans:arg=val
ue
```

The following example applies a transform installed under the name "example" that expects a parameter named "reviewer":

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT \
    -d@./the-body -H "Content-type: application/xml" \
    'http://localhost:8003/v1/documents?uri=/doc/theDoc.xml&transform=
example&trans:reviewer=me'
```

For a complete example, see "XQuery Example: Adding an Attribute During Ingestion" on page 280 or "XSLT Example: Adding an Attribute During Ingestion" on page 282.

3.3 Retrieving Documents from the Database

To a retrieve document from the database, make a GET request to the `/documents` service. You can retrieve just the contents, just the metadata, or both contents and metadata. This section covers the following topics:

- Retrieving the Contents of a Document
- Retrieving Metadata About a Document
- Retrieving Content and Metadata in a Single Request
- Transforming Content During Retrieval

To retrieve multiple documents in a single request, see "Reading and Writing Multiple Documents" on page 206.

3.3.1 Retrieving the Contents of a Document

To retrieve a document from the database, construct a GET request of the following form:

```
http://host:port/version/documents?uri=doc_uri
```

HTTP content type negotiation is not supported. If the HTTP Accept header is not set, MarkLogic Server uses the file extension on the document URI to determine the response content type, based on the server-wide MIME type mapping definitions. See "Mimetypes" in the Admin Interface.

Though content negotiation is not supported, you can use the transform feature to apply server-side transformations to the content before the response is constructed. For details, see "Working With Content Transformations" on page 269.

To retrieve multiple documents in a single request, see "Reading and Writing Multiple Documents" on page 206.

3.3.2 Retrieving Metadata About a Document

To retrieve metadata about a document without retrieving the contents, construct a GET request of the following form:

```
http://host:port/version/documents?uri=doc_uri&category=metadata_category
```

Where `category` can appear multiple times, with the values described in "Metadata Categories" on page 124.

When constructing the request:

1. Set the `category` parameter to the type of metadata to retrieve. Specify `category` multiple times to request more than one type of metadata.

2. Specify the metadata format (XML or JSON) in the HTTP `Accept` header or the `format` parameter.

For details on metadata categories and formats, see "Working with Metadata" on page 124.

Use the `format` parameter or the `Accept` header to specify the format of the metadata. If both `format` and `Accept` are set, `format` takes precedence. If neither `format` nor `Accept` is specified, XML is assumed.

To retrieve metadata as XML, set `format` to "xml" or set the `Accept` header to `application/xml`. To retrieve metadata as JSON, set `format` to "json" or set the `Accept` header to `application/json`.

3.3.3 Retrieving Content and Metadata in a Single Request

To retrieve content and metadata for a document in a single request, construct GET request to a URL of the form:

```
http://host:port/version/documents?uri=doc_uri&category=content&category=metadata_category
```

Where `category` can appear multiple times, with the values described in "Metadata Categories" on page 124.

The request response is a `multipart/mixed` message, with the metadata in the first body part and content in the second body part. The Content-Type headers for the parts are determined as follows:

- The MIME type of the metadata part is determined by the `format` parameter, which you can set to either `xml` or `json`; the default is xml. For details on metadata format, see "Working with Metadata" on page 124.
- The MIME type of the content part is determined by the server-wide MIME type mapping for the document URI extension. See "Mimetypes" in the Admin Interface.

The following example command retrieves a document and its metadata in a single request:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    -H "Accept: multipart/mixed;boundary=BOUNDARY" \
    'http://localhost:8004/v1/documents?uri=/xml/box.xml&category=meta
data&category=content&format=xml'

--BOUNDARY
Content-Type: application/xml
Content-Length: 518

<?xml version="1.0" encoding="UTF-8"?>
<rapi:metadata uri="/xml/box.xml"
    xsi:schemaLocation="http://marklogic.com/rest-api/database
dbmeta.xsd"
    xmlns:rapi="http://marklogic.com/rest-api"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <rapi:collections>
    <rapi:collection>shapes</rapi:collection>
    <rapi:collection>squares</rapi:collection>
  </rapi:collections>
  <rapi:permissions/>
  <prop:properties xmlns:prop="http://marklogic.com/xdmp/property"/>
  <rapi:quality>0</rapi:quality>
</rapi:metadata>
--BOUNDARY
Content-Type: text/xml
Content-Length: 128

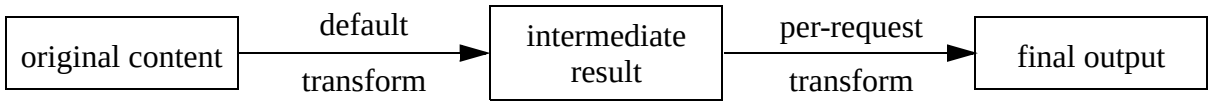
<?xml version="1.0" encoding="UTF-8"?>
<data><mybox>This is my box. There are many like it, but this one is
mine.</mybox></data>
--BOUNDARY--
```

HTTP content negotiation is not supported, but custom server-side content transformations can be applied using the `transform` parameter. For details, see "Retrieving the Contents of a Document" on page 58 and "Working With Content Transformations" on page 269.

3.3.4 Transforming Content During Retrieval

You can apply custom transforms to a document before returning it to the requestor. A transform is an XQuery module or XSLT stylesheet you write and install using `/config/transforms/{name}`. For details, see "Working With Content Transformations" on page 269.

You can configure a default transform that is automatically applied whenever a document is retrieved. You can also specify a per-request transform using the `transform` request parameter. If there is both a default transform and a per-request transform, the transforms are chained together, with the default transform running first. Thus, the output of the default transform is the input to the per-request transform:



To configure a default transformation, set the `document-transform-out` configuration parameter for the MarkLogic REST API instance. Instance-wide parameters are set using `/config/properties`. For details, see "Configuring Instance Properties" on page 41.

To specify a per-request transform, add the `transform` parameter to your request. If the transform expects parameters, specify them using `trans:`*`paramName`* parameters. That is, your request should be of the form:

```
http://host:port/version/v1/documents?...&transform=name&trans:arg=val
ue
```

The following example applies a transform installed under the name "example" that expects a parameter named "reviewer":

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    -H "Accept: application/xml" \
    'http://localhost:8003/v1/documents?uri=/doc/theDoc.xml&transform=
example&trans:reviewer=me'
```

3.4 Partially Updating Document Content or Metadata

This section focuses on modifying a portion of an existing XML or JSON document or its metadata. To learn about inserting or updating an entire document or its metadata, see "Loading Content into the Database" on page 48.

This section covers the following topics:

- Introduction to Content and Metadata Patching
- Basic Steps for Patching Documents
- XML Patch Reference

- Managing XML Namespaces in a Patch
- XML Examples of Partial Updates
- JSON Patch Reference
- JSON Examples of Partial Update
- Patching Metadata
- Constructing Replacement Data on the Server
- How Position Affects the Insertion Point
- Path Expressions Usable in Index Definitions
- Introduction to JSONPath

3.4.1 Introduction to Content and Metadata Patching

A *partial update* is an update you apply to a portion of a document or metadata. For example, inserting an XML element or attribute or changing the value associated with a JSON key. You can only apply partial content updates to XML and JSON documents. You can apply partial metadata updates to any document type.

A *patch* is a partial update descriptor, expressed in XML or JSON, that tells MarkLogic Server where to apply an update and what update to apply. A patch is a wrapper XML element or JSON object that encapsulates one or more update operations.

Use a partial update to do the following operations:

- Add, replace, or delete an XML element, XML attribute, or JSON sub-object of an existing document.
- Add, replace, or delete a subset of the metadata of an existing document. For example, modify a permission or insert a property.
- Dynamically generate replacement content or metadata on MarkLogic Server using builtin or user-defined functions. For details, see “Constructing Replacement Data on the Server” on page 100.

To perform a partial update, send a POST or PATCH request to the `/documents` service with a patch in the request body. For example, the following patch descriptor inserts a new `<child/>` element as the last child of the element located through the XPath expression `/data`:

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:insert context="/data" position="last-child">
    <child>the last one</child>
  </rapi:insert>
</rapi:patch>
```

Assuming the above patch is saved to the file `patch.xml`, the following command uses a POST request to apply the patch to the document with the URI `/doc/example.xml`.

```
$ curl --anyauth --user user:password -X POST -d @./patch.xml -i \
    -H "Content-type: application/xml" \
    -H "X-HTTP-Method-Override: PATCH" \
    http://localhost:8003/v1/documents?uri=/doc/example.xml
```

The following example is an equivalent request using PATCH instead of POST:

```
$ curl --anyauth --user user:password -X PATCH -d @./patch.xml -i \
    -H "Content-type: application/xml" \
    http://localhost:8003/v1/documents?uri=/doc/example.xml
```

For details, see “Basic Steps for Patching Documents” on page 63.

You can apply multiple updates in a single request. You can patch both content and metadata in a single request.

Content negotiation is not supported: A content patch must match the content type of the target document. You cannot submit a JSON patch for an XML document or vice versa.

If a patch contains multiple operations, they are applied independently to the target document. That is, within the same patch, one operation does not affect the `context` or `select` results or the content changes of another. Each operation in a patch is applied independently to every matched node. If any operation in a patch fails with an error, the entire patch fails.

Content transformations are not directly supported in a partial update. However, you can implement a custom replacement content generation function to achieve the same effect. For details, see “Constructing Replacement Data on the Server” on page 100.

3.4.2 Basic Steps for Patching Documents

To perform a partial update of an XML or JSON document, or of document metadata, send a POST or PATCH request to the `/documents` service with a URL of the following form:

```
http://host:port/version/documents?uri=doc_uri
```

The body of the request must contain a content patch, as described in “XML Patch Reference” on page 64 and “JSON Patch Reference” on page 82. You can patch metadata and content in the same request by using the category request parameter; for details, see “Patching Metadata” on page 98.

When constructing your request, follow these guidelines:

1. Set the `uri` parameter to the URI of the target document in the database.

2. Set the `category` request parameter to `content`, `metadata`, and/or one of the metadata subcategories. The default is `content`.

3. Set the `Content-type` to either `application/xml` or `application/json`. When patching content, you must use the content type that matches the target document; only XML and JSON are supported.

4. Place the XML or JSON patch in the request body.

5. If you POST, set the request header `X-HTTP-Method-Override` to `PATCH` to tell MarkLogic Server this is a partial update request.

New content that is added or replaced can be specified directly in the patch, or generated on MarkLogic Server by a replacement content generator function. For details, see "Constructing Replacement Data on the Server" on page 100.

3.4.3 XML Patch Reference

This section summarizes the structure of the XML patch structure used to describe partial updates to XML documents and document metatdata. Each section includes a syntax summary, a description of the components, and an example.

The following elements, all in the namespace http://marklogic.com/rest-api, are covered. Start wit the `<patch/>` wrapper.

- patch
- insert
- replace
- replace-insert
- delete
- replace-library

3.4.3.1 patch

The top level wrapper element around a partial update descriptor. A `<patch/>` has the following structure:

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:insert />
  <rapi:replace />
  <rapi:replace-insert />
  <rapi:delete />
  <rapi:replace-library />
</rapi:patch>
```

All elements are optional. The operations can occur multiple times. The `<replace-library/>` can occur at most once and is only required if you use user-defined functions to generate replacement content server-side; for details, see "Constructing Replacement Data on the Server" on page 100.

For example, the following patch includes an insertion, deletion, and replacement:

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:insert context="/inventory/history" position="last-child">
    <modified>2012-11-5</modified>
  </rapi:insert>
  <rapi:delete select="saleExpirationDate"/>
  <rapi:replace select="price" apply="ml.multiply">1.1</rapi:replace>
</rapi:patch>
```

Where a patch operation includes new XML content, such as an element insertion, the new content must use namespaces appropriate to the target document. Namespaces declared in the root `<patch/>` node are in scope for this content. For details, see “Managing XML Namespaces in a Patch” on page 74.

For more details, see the individual operations.

3.4.3.2 insert

Insert a new element, attribute, text node, comment, or processing instruction in an XML document or in document metadata. An `<insert/>` element has the following structure:

```
<insert context=xpath-expr position=pos-selector
        cardinality=occurrence>
  content-to-insert
</insert>
```

For example, the following patch adds a `<modified/>` element as the last child of every `<history/>` node that matches the XPath expression `/inventory/history`:

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:insert context="/inventory/history" position="last-child">
    <modified>2012-11-5</modified>
  </rapi:insert>
</rapi:patch>
```

For additional examples, see “XML Examples of Partial Updates” on page 75.

The following table summarizes the parts of an `<insert/>` element.

Component	Req'd	Description
`@context=`*`xpath-expr`*	Y	An XPath expression that selects an existing node(s) or attribute on which to operate. Namespaces declared on the root `<patch/>` node are in scope for this path. When inserting an attribute, `@context` can identify either the containing element(s) or another attribute of the containing element(s). In all other cases, `@context` should identify one or more nodes. If no matches are found for the `@context` expression, the operation is silently ignored. The path expression is restricted to the subset of XPath that can be used to define an index. For details, see "Path Expressions Usable in Index Definitions" on page 109.
`@position=`*`pos-selector`*	Y	Where to insert the content, relative to the node(s) selected by `@context`. The attribute value must be one of `before`, `after`, or `last-child`.

Component	Req'd	Description
`@cardinality=`*`occurrence`*	N	The required occurrence of matches to `@position`. If the number of matches does not meet the expectation, the operation fails and an error is returned. Allowed values: • Zero or one matches required: `?` (question mark) • Exactly one match required: `.` (period) • Zero or more matches required: `*` (asterisk) • One or more matches required: `+` (plus) Default: * (The occurrence requirement is always met).
`content-to-insert`	Y	When inserting elements, text, comments, and processing directives, specify the new nodes as they should appear in the target document. When inserting attributes, specify the attribute(s) on a `<rapi:attribute-list/>` element. The following example specifies 2 attributes to insert: `<rapi:attribute-list a1="v1" a2="v2"/>`

3.4.3.3 replace

Replace an existing element or attribute. If no matching element or attribute exists, the operation is silently ignored. A `<replace/>` element has the following structure:

```
<replace select=xpath-expr cardinality=occurrence apply=func-name>
  replacement-content
</replace>
```

For example, the following patch replaces the first `<child/>` element of `<parent/>`:

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:replace select="/parent/child[1]">
    <child>REPLACED</child>
  </rapi:replace>
</rapi:patch>
```

For additional examples, see "XML Examples of Partial Updates" on page 75.

You can also supply a content generation builtin or user-defined function using `@apply`, using the format below. For example, you can use the builtin `ml.multiply` function to multiply the current value of an element that contains numeric data, without needing to know the current value.

```
<rapi:replace select="my-elem" apply=ml.multiply>5</rapi:replace>
```

For details, see “Constructing Replacement Data on the Server” on page 100.

The following table summarizes the parts of a `<replace/>` element.

Component	Req’d	Description
`@select=`*`xpath-expr`*	Y	An XPath expression that selects an existing node(s) or attribute to replace. Namespaces declared on the root `<patch/>` node are in scope for this path. The path expression is restricted to the subset of XPath that can be used to define an index. For details, see “Path Expressions Usable in Index Definitions” on page 109. The selected node cannot be the target of any other operation in the patch. The ancestor of the selected node may not be modified by a `delete`, `replace`, or `replace-insert` operation in the same patch.
`@cardinality=`*`occurrence`*	N	The required occurrence of matches to `@select`. If the number of matches does not meet the expectation, the operation fails and an error is returned. Allowed values: • Zero or one matches required: `?` (question mark) • Exactly one match required: `.` (period) • Zero or more matches required: `*` (asterisk) • One or more matches required: `+` (plus) Default: * (The occurrence requirement is always met).

Component	Req'd	Description

replacement-content	N	The content with which to replace the selected node or attribute. If present, this must be a single node. If there is no *replacement-content*, you must specify a content generation function using `@apply`.
		
		To replace an element, specify either a replacement element or text. When you use text, the text replaces just the content of the target element.
		
		To replace an element with a text node, wrap the text in a `<text/>` element in the namespace `http://marklogic.com/rest-api`. For example:
		
		`<rapi:text>replacement text</rapi:text>`
		
		To replace an attribute, use either a `<text/>` or `<attribute-list/>` wrapper in the namespace `http://marklogic.com/rest-api`. For example:
		
		`<rapi:attribute-list my-attr="`*new-value*`" />`
		`<rapi:text>`*new-value*`</rapi:text>`

`@apply=`*func-name*	N	The localname of a replacement content generation function. If you do not specify a function, the operation must include *replacement-content*. If you name a user-defined function, the `<patch/>` must also contain a `<replace-library/>` element. For details, see "Constructing Replacement Data on the Server" on page 100.

3.4.3.4 replace-insert

Replace an element or attribute; if there are no existing matching elements or attributes, perform an insertion operation instead. A `<replace-insert/>` element has the following structure:

```
<replace-insert select=xpath-expr context=xpath-expr position=pos
    cardinality=occurrence apply=func-name>
  replacement-content
</replace-insert>
```

For example, the following patch replaces the first `<grandchild/>` element of a `<child/>` of `<parent/>` elements. If no `<child/>` element has a `<grandchild/>` element, one is inserted in every `<child/>` of a `<parent/>` element.

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:replace-insert context="/parent/child" select="grandchild[1]"
      position="last-child">
    <grandchild>CHANGED</grandchild>
  </rapi:replace-insert>
</rapi:patch>
```

For additional examples, see “XML Examples of Partial Updates” on page 75.

You can also use `@apply` to specify a content generation builtin or user-defined function for generating dynamic content. For details, see “Constructing Replacement Data on the Server” on page 100.

The following table summarizes the parts of a `<replace-insert/>` element.

Component	Req’d	Description
`@select=`*`xpath-expr`*	Y	An XPath expression that selects an existing node(s) or attribute to replace. Namespaces declared on the root `<patch/>` node are in scope for this path. The path may be absolute or relative to the `@context` path. The path expression is restricted to the subset of XPath that can be used to define an index. For details, see “Path Expressions Usable in Index Definitions” on page 109. If no matches are found for the `@select` expression, `@context` and `@position` are used to attempt an insert. If no match is then found for `@context`, the behavior depends on the value of `@cardinality`. The selected node cannot be the target of any other operation in the patch. The ancestor of the selected node may not be modified by a delete, replace, or replace-insert operation in the same patch.

Component	Req'd	Description
replacement-content	N	The content with which to replace the selected node or attribute. If there is no *replacement-content*, you must specify a content generation function using @apply. To replace an element, specify either a replacement element or text. When you use text, the text replaces just the content of the target element. To replace an element with a text node, wrap the text in a <text/> element in the namespace http://marklogic.com/rest-api. For example: <rapi:text>replacement text</rapi:text> To replace an attribute, use either a <text/> or <attribute-list/> wrapper in the namespace http://marklogic.com/rest-api. For example: <rapi:attribute-list my-attr="*new-value*" /> <rapi:text>*new-value*</rapi:text>
@context=*xpath-expr*	Y	An XPath expression that selects an existing node(s) or attribute on which to operate if the @select target does not exist. @context is also used to resolve @select if the select expression is a relative path. Namespaces declared on the root <patch/> node are in scope for this path. The path expression is restricted to the subset of XPath that can be used to define an index. That is, path expressions for which cts:valid-index-path returns true. When inserting an attribute, @context can identify either the containing element(s) or another attribute of the containing element(s). In all other cases, @context should identify one or more nodes. If no matches are found for either the @context or @select expressions, the operation is silently ignored. The ancestor of the selected node may not be modified by a delete, replace, or replace-insert operation in the same patch.

Component	Req'd	Description
`@position=`*`pos-selector`*	Y	If `@select` does not match anything, where to insert the content, relative to the node(s) selected by `@context`. The attribute value must be one of `before`, `after`, or `last-child`. Ignored for attributes.
`@cardinality=`*`occurrence`*	N	The required occurrence of matches to `@position`. If the number of matches does not meet the expectation, the operation fails and an error is returned. Allowed values: • Zero or one matches required: `?` (question mark) • Exactly one match required: `.` (period) • Zero or more matches required: `*` (asterisk) • One or more matches required: `+` (plus) Default: * (The occurrence requirement is always met).
`@apply=`*`func-name`*	N	The localname of a replacement content generation function. If you do not specify a function, the operation must include *`replacement-content`*. If you name a user-defined function, the `<patch/>` must also contain a `<replace-library/>` element. For details, see "Constructing Replacement Data on the Server" on page 100.

3.4.3.5 delete

Remove an element or attribute from an existing node. A `<delete/>` element has the following structure:

```
<delete select=xpath-expr cardinality=occurrence />
```

For example, the following patch deletes the first `<child/>` element of a `<parent/>` and removes the `my-attr` attribute from the `<parent/>` element.

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:delete select="/parent/child[1]" />
  <rapi:delete select="/parent/@my-attr" />
</rapi:patch>
```

For additional examples, see "XML Examples of Partial Updates" on page 75.

The following table summarizes the parts of a `<delete/>` element.

Component	Req'd	Description
`@select=`*`xpath-expr`*	Y	An XPath expression that selects the node(s) or attribute(s) to delete. Namespaces declared on the root `<patch/>` node are in scope for this path. The path expression is restricted to the subset of XPath that can be used to define an index. For details, see "Path Expressions Usable in Index Definitions" on page 109. The selected node cannot be the target of any other operation in the patch. The ancestor of the selected node may not be modified by a delete, replace, or replace-insert operation in the same patch.
`@cardinality=`*`occurrence`*	N	The required occurrence of matches to `@select`. If the number of matches does not meet the expectation, the operation fails and an error is returned. Allowed values: • Zero or one matches required: `?` (question mark) • Exactly one match required: `.` (period) • Zero or more matches required: `*` (asterisk) • One or more matches required: `+` (plus) Default: * (The occurrence requirement is always met).

3.4.3.6 replace-library

Specify an XQuery library module that contains user-defined replacement content generation functions. These functions can be used in `@apply` on `<replace/>` and `<replace-insert/>` operations. For details, see "Constructing Replacement Data on the Server" on page 100.

For example, the following patch uses the function `my-ns:my-func` in the library module in the with the URI `/my.domain/my-module.xqy`.

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:replace-library
    @at="/my.domain/my-module.xqy" @ns="my-ns" />
  <rapi:replace select="/parent/child[1]" @apply="my-func" />
</rapi:patch>
```

For additional examples, see "Constructing Replacement Data on the Server" on page 100.

The following table summarizes the parts of a `<replace-library/>` element.

Component	Req'd	Description
`@at=`*`lib-module-path`*	N	The path to the XQuery library module containing user-defined replacement content generation functions. The module must be installed in the modules database associated with the REST API instance.
`@ns=`*`func-namespace`*	N	The module namespace alias defined by the `@at` module.

3.4.4 Managing XML Namespaces in a Patch

The patch descriptor must be in the namespace `http://marklogic.com/rest-api`. To distinguish the `<patch/>` elements from elements in your document, you should include an alias for the patch descriptor namespace. For example:

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
    ...
</rapi:patch>
```

Declare additional namespace aliases for the content described by the patch. Namespaces declared on the `<patch/>` root element are in scope in `@context` and `@select` path expressions and in the content portion of `<insert/>`, `<replace/>` and `<replace-insert/>` elements.

For example, if you patch a document with the following contents:

```
<parent xmlns="http://my/namespace">
  <child>first</child>
</parent>
```

Then your `<patch/>` should either define `http://my/namespace` as the default namespace or define a namespace alias for it. For example, the following patch inserts a `<child/>` element when you define a default namespace:

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api"
    xmlns="http://my/namespace">
  <rapi:insert context="/parent" position="last-child">
    <child>last</child>
  </rapi:insert>
</rapi:patch>
```

The following is the same patch, using an explicit namespace alias of `my-ns`:

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api"
    xmlns:my-ns="http://my/namespace">
  <rapi:insert context="/my-ns:parent" position="last-child">
```

```
    <my-ns:child>last</my-ns:child>
  </rapi:insert>
</rapi:patch>
```

Failure to use namespace declarations properly in a patch can lead to surprising results. For example, if you did not include a namespace declaration for `http://my/namespace`, the patch would not work because the context XPath expression would not match any elements in the target document. Similarly, if you include a namespace alias but do not use it in the definition of the new child, the new child element would be inserted in no namespace: `<child xmlns="">...</child>`.

3.4.5 XML Examples of Partial Updates

This section includes the following examples of applying a partial update to an XML document or metadata:

- Example cURL Commands
- Example: Inserting an Element
- Example: Inserting an Attribute
- Example: Inserting a Text Node
- Example: Inserting a Comment or Processing Instruction
- Example: Multiple Inserts in One Patch
- Example: Replacing an Element or Element Contents
- Example: Replacing an Attribute Value
- Example: Replacing or Inserting an Element in One Operation
- Example: Replacing or Inserting an Attribute in One Operation
- Example: Deleting an Element or Attribute

Note: Many of the examples include extra whitespace to improve readability. You should not do likewise unless you want extra whitespace in your target document. Whitespace in the body of an `<insert/>`, `<replace/>`, or `<replace-insert/>` operation is usually significant.

3.4.5.1 Example cURL Commands

The examples in this section can be applied using commands similar to the following, assuming you copy the sample input document to a file called `example.xml` and the patch to a file called `patch.xml`.

```
# Windows users, see Modifying the Example Commands for Windows
# Create the example document
$ curl --anyauth --user user:password -X PUT -d @./example.xml \
    -i -H "Content-type: application/xml" \
    http://localhost:8003/v1/documents?uri=/docs/example.xml
```

```
# Apply the patch
$ curl --anyauth --user user:password -X POST -d @./patch.xml \
    -i -H "Content-type: application/xml" \
    -H "X-HTTP-Method-Override: PATCH" \
    http://localhost:8003/v1/documents?uri=/docs/example.xml

# Retrieve the results
curl --anyauth --user user:password -X GET \
    -i -H "Accept: application/xml" \
    http://localhost:8003/v1/documents?uri=/docs/example.xml
```

3.4.5.2 Example: Inserting an Element

The following patch inserts an `<INSERTED/>` element before each `<child/>` element matched by the context expression.

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:insert context="/parent/child" position="before">
    <INSERTED/>
  </rapi:insert>
</rapi:patch>
```

The table below shows how applying the patch changes the target document.

Before Update	After Update

`<parent>`
`  <child>one</child>`
`  <child>two</child>`
`  <child>three</child>`
`</parent>`
	`<parent>`
	`  **<INSERTED/>**`
	`  <child>one</child>`
	`  **<INSERTED/>**`
	`  <child>two</child>`
	`  **<INSERTED/>**`
	`  <child>three</child>`
	`</parent>`

3.4.5.3 Example: Inserting an Attribute

The following patch adds two attributes, `my-attr1` and `my-attr2`, on the second `<child/>` element.

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:insert context="/parent/child[2]">
    <rapi:attribute-list my-attr1="val1" my-attr2="val2" />
  </rapi:insert>
</rapi:patch>
```

The table below shows how applying the patch changes the target document.

Before Update	After Update
<parent> <child>one</child> <child>two</child> <child>three</child> </parent>	<parent> <child>one</child> <child **my-attr1="val1" my-attr2="val2"**> two </child> <child>three</child> </parent>

3.4.5.4 Example: Inserting a Text Node

The following patch inserts a text node before the second `<child/>` element matched by the context expression.

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:insert context="/parent/child[2]"
     position="before">INSERTED</rapi:insert>
</rapi:patch>
```

The table below shows how applying the patch changes the target document.

Before Update	After Update
<parent> <child>one</child> <child>two</child> <child>three</child> </parent>	<parent> <child>one</child> **INSERTED** <child>two</child> <child>three</child> </parent>

3.4.5.5 Example: Inserting a Comment or Processing Instruction

The following patch inserts a comment as the first child of every `<parent/>` element matched by the context expression.

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:insert context="/parent/*[1]" position="before">
    <!-- INSERTED -->
  </rapi:insert>
</rapi:patch>
```

The table below shows how applying the patch changes the target document.

Before Update	After Update
<parent> <child>one</child> <child>two</child> <child>three</child> </parent>	<parent> **<!-- INSERTED -->** <child>one</child> <child>two</child> <child>three</child> </parent>

3.4.5.6 Example: Multiple Inserts in One Patch

The following patch performs several insert operations on the target document:

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:insert context="/parent/child" position="before">
    <INSERTED/>
  </rapi:insert>
  <rapi:insert context="/parent/child[2]"
      position="before">INSERTED</rapi:insert>
  <rapi:insert context="/parent/child[2]">
    <rapi:attribute-list my-attr="val" />
  </rapi:insert>
  <rapi:insert context="/parent/*[1]" position="before">
    <!-- INSERTED -->
  </rapi:insert>
</rapi:patch>
```

The table below shows how applying the patch changes the target document. Notice that the insert operations act independently on the target document, rather than interacting with each other. For example, the comment insertion operation, which uses the context expression `/parent/*[1]`, inserts the comment before the first original `<child/>` element, not before the `<INSERTED/>` element added by one of the other operations in the patch.

Before Update	After Update

<parent>
  <child>one</child>
  <child>two</child>
  <child>three</child>
</parent>
	<parent>
	  <INSERTED/>
	  <!-- INSERTED -->
	  <child>one</child>
	  <INSERTED/>
	  INSERTED
	  <child **my-attr="val"**>two</child>
	  <INSERTED/>
	  <child>three</child>
	</parent>

3.4.5.7 Example: Replacing an Element or Element Contents

The following patch replaces the first `<child/>` node with a new element, replaces the contents of the second `<child/>`, and replaces the third `<child/>` with a text node.

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <!-- replace an entire element -->
  <rapi:replace select="/parent/child[1]">
    <REPLACED />
  </rapi:replace>

  <!-- replace the element contents -->
  <rapi:replace select="/parent/child[2]">
    REPLACED
  </rapi:replace>

  <!-- replace an element with a text node -->
  <rapi:replace select="/parent/child[3]">
    <rapi:text>REPLACED</rapi:text>
  </rapi:replace>
</rapi:patch>
```

The table below shows how applying the patch changes the target document.

Before Update	After Update

`<parent>`
`  <child>one</child>`
`  <child>two</child>`
`  <child>three</child>`
`</parent>`
	`<parent>`
	`  **<REPLACED />**`
	`  <child>**REPLACED**</child>`
	`  **REPLACED**`
	`</parent>`

3.4.5.8 Example: Replacing an Attribute Value

The following patch demonstrates two methods of replacing the value of an attribute. The first `<replace/>` uses a `<text/>` wrapper and the second uses an `<attribute-list>` container element.

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:replace select="/parent/@attr1">
    <rapi:text>REPLACE-1</rapi:text>
  </rapi:replace>
  <rapi:replace select="/parent/child/@attr2">
    <rapi:attribute-list attr2="REPLACE-2"/>
  </rapi:replace>
</rapi:patch>
```

The table below shows how applying the patch changes the target document.

Before Update	After Update

`<parent attr1="val1">`
`  <child attr2="val2">one</child>`
`  <child attr2="val2">two</child>`
`</parent>`
	`<parent attr1="`**`REPLACE-1`**`">`
	`  <child attr2="`**`REPLACE-2`**`">one</child>`
	`  <child attr2="`**`REPLACE-2`**`">two</child>`
	`</parent>`

3.4.5.9 Example: Replacing or Inserting an Element in One Operation

The following patch replaces the first `<grandchild/>` element of a `<child/>` of `<parent/>` elements. If no `<child/>` element has a `<grandchild/>` element, one is inserted in every `<child/>` of a `<parent/>` element.

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:replace-insert context="/parent/child" select="grandchild[1]"
    position="last-child">
    <grandchild>CHANGED</grandchild>
  </rapi:replace-insert>
</rapi:patch>
```

The following table illustrates applying this patch to two documents, one for which there is a match to the replacement target, and one for which there is no match. In the first case, the node matching the `@select` expression (relative to `@context`) is replaced with the patch contents. In the second case, there are no matching nodes, so the content in the patch is inserted into all nodes that match the `@context` expression.

Before Update	After Update

Document contains matching elements.
 	
`<parent>`
`  <child/>`
`  <child>`
`  <grandchild/>`
`  <grandchild>`
`  unchanged`
`  </grandchild>`
`  </child>`
`</parent>`
	Matching elements replaced.
	
	`<parent>`
	`  <child/>`
	`  <child>`
	`  **<grandchild>CHANGED</grandchild>**`
	`  <grandchild>`
	`  unchanged`
	`  </grandchild>`
	`  </child>`
	`</parent>`

Document contains no matching elements.
 	
`<parent>`
`  <child/>`
`  <child/>`
`</parent>`
	New elements inserted using `@context`.
	
	`<parent>`
	`  <child>`
	`  **<grandchild>CHANGED</grandchild>**`
	`  </child>`
	`  <child>`
	`  **<grandchild>CHANGED</grandchild>**`
	`  </child>`
	`</parent>`

The following patch is equivalent, except that the `<grandchild/>` elements are replaced with a text node:

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:replace-insert context="/parent/child" select="grandchild[1]"
      position="last-child">
    <rapi:text>CHANGED</rapi:text>
  </rapi:replace-insert>
</rapi:patch>
```

3.4.5.10 Example: Replacing or Inserting an Attribute in One Operation

The following patch replaces `/parent/child/@my-attr` and inserts `/parent/child/@new-attr`.

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:replace-insert select="@my-attr" context="/parent/child">
    REPLACED
```

```
    </rapi:replace-insert>
    <rapi:replace-insert select="@new-attr" context="/parent/child">
      <rapi:attribute-list new-attr="INSERTED" />
    </rapi:replace-insert>
  </rapi:patch>
```

The following table illustrates the results of applying this patch:

Before Update	After Update
<parent> <child /> <child my-attr="val2" /> <child /> </parent>	<parent> <child **new-attr="INSERTED"**/> <child **my-attr="REPLACED"** **new-attr="INSERTED"**/> <child **new-attr="INSERTED"**/> </parent>

3.4.5.11 Example: Deleting an Element or Attribute

The following patch deletes all `<grandchild/>` elements and deletes an attribute from the `<parent/>` element.

```
  <rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
    <rapi:delete select="/parent/child/grandchild"/>
    <rapi:delete select="/parent/@my-attr"/>
  </rapi:patch>
```

The table below shows how applying the patch changes the target document.

Before Update	After Update
<parent my-attr="val"> <child/> <child> <grandchild/> <grandchild/> </child> </parent>	<parent> <child/> <child></child> </parent>

3.4.6 JSON Patch Reference

This section summarizes the structure of the JSON patch structured used to describe partial updates to JSON documents. Each section includes a syntax summary, a description of component key-value pairs, and an example.

The following key-value pairs are covered:

- patch
- insert
- replace
- replace-insert
- delete
- replace-library

3.4.6.1 patch

A patch is the top level wrapper object for applying partial updates to JSON documents. A JSON patch has the following structure. A patch object can contain multiple operations.

```
{"patch": [
    insert,
    replace,
    replace-insert,
    delete,
    replace-library
] }
```

All subobjects are optional. The operations (`insert`, `replace`, `replace-insert`) can occur multiple times. You can only have one `replace-library`, and it is only required if you use user-defined functions to generate replacement content.

For example, the following patch describes an insert and a replace operation. For more examples, see “JSON Examples of Partial Update” on page 92.

```
{"patch": [
  { "insert": {
      "context": "$.parent.child",
      "position": "last-child",
      "content": { "new-key": "new-value" }
  }},
  { "replace": {
      "select": "$.parent.child[0]",
      "content": "new-value"
    }}
]}
```

3.4.6.2 insert

Insert new objects or array elements. An `insert` operation has the following structure:

```
{
  "insert": {
    "context": jsonpath-expr,
    "position": pos-selector,
    "content": new-content,
    "cardinality": occurrence
  }
}
```

For example, the following patch inserts the value "INSERTED" after any array element

```
{"insert": {
    "context": "$parent.child.[*][?(@='insertion-point')]",
    "position": "after",
    "content": "INSERTED"
}}
```

The following table summarizes the components of an `insert` operation.

Key-Value Pair	Req'd	Description
`"context": jsonpath-expr`	Y	A JSONPath expression that selects an existing key-value pair or array element on which to operate. The expression can select multiple items. If no matches are found for the `context` expression, the operation is silently ignored. The path expression is restricted to the JSONPath equivalent of the subset of XPath that can be used to define an index. For details, see "Path Expressions Usable in Index Definitions" on page 109.
`"position" : pos-selector`	Y	Where to insert the content, relative to the key-value pair or value selected by `context`. The *pos-selector* must be one of `"before"`, `"after"`, or `"last-child"`. For details, see "Specifying Position in JSON" on page 108.

Key-Value Pair	Req'd	Description
`"content": new-content`	Y	The new content to be inserted.
`cardinality: occurrence`	N	The required occurrence of matches to `position`. If the number of matches does not meet the expectation, the operation fails and an error is returned. Allowed values: • Zero or one matches required: `?` (question mark) • Exactly one match required: `.` (period) • Zero or more matches required: `*` (asterisk) • One or more matches required: `+` (plus) Default: * (The occurrence requirement is always met).

3.4.6.3 replace

Replace an existing key-value pair or array element. If no matching key-value pair or array element exists, the operation is silently ignored. A replace operation has the following structure:

```
{ "replace": {
    "select" : jsonpath-expr,
    "content" : new-value,
    "cardinality": occurrence,
    "apply" : func-name
  }
}
```

For example, the following patch replaces the value of the first array element in the key-value pair with key "child" that is a subobject of the key-value pair with key "parent":

```
{"patch": [
  {"replace": {
      "select": "$.parent.child[0]",
      "content": "REPLACED"
  }}
] }
```

For additional examples, see "JSON Examples of Partial Update" on page 92.

You can use `apply` to specify a content generation builtin or user-defined function for generating dynamic content. For details, see "Constructing Replacement Data on the Server" on page 100.

The following table summarizes the parts of a `replace` operation.

Component	Req'd	Description
`"select": xpath-expr`	Y	A JSONPath expression that selects the object or array element to replace. If no matches are found for the `select` expression, the operation is silently ignored. The path expression is restricted to the JSONPath equivalent of the subset of XPath that can be used to define an index. For details, see "Path Expressions Usable in Index Definitions" on page 109. The selected item(s) cannot be the target of any other operation in the patch. The ancestor of the selected item may not be modified by a `delete`, `replace`, or `replace-insert` operation in the same patch.
`"content" : new-value`	N	The replacement value. If there is no *`new-value`*, you must specify a content generation function using `apply`. If `select` targets a key-value pair and *`new-value`* is an object, the operation replaces the entire target key-value pair. Otherwise, the operation replaces just the value.
`cardinality: occurrence`	N	The required occurrence of matches to `select`. If the number of matches does not meet the expectation, the operation fails and an error is returned. Allowed values: • Zero or one matches required: `?` (question mark) • Exactly one match required: `.` (period) • Zero or more matches required: `*` (asterisk) • One or more matches required: `+` (plus) Default: * (The occurrence requirement is always met).

Component	Req'd	Description
`"apply": func-name`	N	The localname of a replacement content generation function. If you do not specify a function, the operation must include a `content` key-value pair. If you name a user-defined function, the `patch` must include a `replace-library` key-value pair that describes the XQuery library module containing the function implementation. For details, see "Constructing Replacement Data on the Server" on page 100.

3.4.6.4 replace-insert

Replace an element or attribute; if there are no existing matching elements or attributes, perform an insertion operation instead. A `replace-insert` operation has the following structure:

```
{ "replace-insert": {
    "select" : jsonpath-expr,
    "context" : jsonpath-expr,
    "position": pos-selector,
    "cardinality": occurrence,
    "content" : new-content,
    "apply" : func-name
  }
}
```

Optionally, you can supply a builtin or user-defined content generation function name using `apply`. If the function does not expect parameters, you can omit content when using `apply`. For details, see "Constructing Replacement Data on the Server" on page 100.

For example, the following patch replaces the value of `"target"` with `"new-value"` in the array value of `$.parent.child`, if it exists. If no such value is found, then `"new-value"` is inserted as the last element in the array addressable as `$.parent.child`.

```
{ "patch": [
    { "replace-insert" : {
        "select": "['target']",
        "context": "$.parent.child",
        "position": "last-child",
        "content": {"new-value"}
    }}
]}
```

For additional examples, see "JSON Examples of Partial Update" on page 92.

You can also use `apply` to specify a content generation builtin or user-defined function for generating dynamic content. For details, see "Constructing Replacement Data on the Server" on page 100.

The following table summarizes the parts of a `replace-insert` operation.

Component	Req'd	Description
`"select"` : *xpath-expr*	Y	A JSONPath expression that selects the object or array element to replace. If no matches are found for the `select` expression, `context` and `position` are used to attempt an insert. If no match is found for `context`, the operation does nothing. The path expression is restricted to the JSONPath equivalent of the subset of XPath that can be used to define an index. For details, see "Path Expressions Usable in Index Definitions" on page 109. The selected item(s) cannot be the target of any other operation in the patch. The ancestor of the selected item may not be modified by a `delete`, `replace`, or `replace-insert` operation in the same patch.
new-content	N	The content with which to replace the selected value. If there is no *`replacement-content`*, you must specify a content generation function using `apply`. If `select` targets a key-value pair and *`new-content`* is an object, the operation replaces the entire target key-value pair. Otherwise, the operation replaces just the value.

Component	Req'd	Description
`"context"` : *xpath-expr*	Y	A JSONPath expression that selects an existing key-value pair or array element on which to operate. The expression can select multiple items. If no matches are found for the either the `select` or `context` expression, the operation is silently ignored. The path expression is restricted to the JSONPath equivalent of the subset of XPath that can be used to define an index. For details, see "Path Expressions Usable in Index Definitions" on page 109. The ancestor of the selected node may not be modified by a `delete`, `replace`, or `replace-insert` operation in the same patch.
`"position"` : *pos-selector*	N	If `select` does not match anything, where to insert the content, relative to the key-value pair or value selected by `context`. The *pos-selector* must be one of `"before"`, `"after"`, or `"last-child"`. For details, see "Specifying Position in JSON" on page 108. Default: `last-child`.
`cardinality`: *occurrence*	N	The required occurrence of matches to `position`. If the number of matches does not meet the expectation, the operation fails and an error is returned. Allowed values: • Zero or one matches required: `?` (question mark) • Exactly one match required: `.` (period) • Zero or more matches required: `*` (asterisk) • One or more matches required: `+` (plus) Default: * (The occurrence requirement is always met).

Component	Req'd	Description
`"apply"` : *func-name*	N	The localname of a replacement content generation function. If you do not specify a function, the operation must include a `content` key-value pair. If you name a user-defined function, the `patch` must include a `replace-library` key-value pair that describes the XQuery library module containing the function implementation. For details, see "Constructing Replacement Data on the Server" on page 100.

3.4.6.5 delete

Remove a key-value pair or array element. A `delete` operation has the following structure:

```
{ "delete": {
    "select" : jsonpath-expr,
    "cardinality": occurrence
} }
```

For example, the following patch deletes any key-value pair with key `child` that is a subobject of a key-value pair with key `parent`.

```
{
  "patch": [
    { "delete" : { "select": "$.parent.child" } }
  ]
}
```

For additional examples, see "JSON Examples of Partial Update" on page 92.

The following table summarizes the parts of a `delete` operation.

Component	Req'd	Description
`select : jsonpath-expr`	Y	A JSONPath expression that selects the key-value pair or array element to remove. If no matches are found for the `select` expression, the operation is silently ignored. The path expression is restricted to the JSONPath equivalent of the subset of XPath that can be used to define an index. For details, see "Path Expressions Usable in Index Definitions" on page 109. The selected item(s) cannot be the target of any other operation in the patch. The ancestor of the selected item may not be modified by a `delete`, `replace`, or `replace-insert` operation in the same patch.
`cardinality: occurrence`	N	The required occurrence of matches to `select`. If the number of matches does not meet the expectation, the operation fails and an error is returned. Allowed values: • Zero or one matches required: `?` (question mark) • Exactly one match required: `.` (period) • Zero or more matches required: `*` (asterisk) • One or more matches required: `+` (plus) Default: * (The occurrence requirement is always met).

3.4.6.6 replace-library

Specify an XQuery library module that contains user-defined replacement content generation functions. These functions can be used in the `apply` key-value pair of a `replace` or `replace-insert` operation. A patch can contain at most one `replace-library`. For details, see "Constructing Replacement Data on the Server" on page 100.

For example, the following patch uses the function `my-ns:my-func` in the library module in the with the URI `/my.domain/my-module.xqy`.

```
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:replace-library
    @at="/my.domain/my-module.xqy" @ns="my-ns" />
  <rapi:replace select="/parent/child[1]" @apply="my-func" />
</rapi:patch>
```

For additional examples, see “Constructing Replacement Data on the Server” on page 100.

The following table summarizes the parts of a `<replace-library/>` element.

Component	Req’d	Description
`"at": lib-module-path`	N	The path to the XQuery library module containing user-defined replacement content generation functions. The module must be installed in the modules database associated with the REST API instance.
`"ns": func-namespace`	N	The module namespace alias defined by the `at` module.

3.4.7 JSON Examples of Partial Update

This section includes the examples of applying a partial update with a JSON patch. The following topics are covered:

- Example cURL Commands
- Example: Insert
- Example: Replace
- Example: Replace-Insert
- Example: Delete

3.4.7.1 Example cURL Commands

The examples in this section can be applied using commands similar to the following, assuming you copy the sample input document to a file called `example.json` and the patch to a file called `patch.json`.

```
# Windows users, see Modifying the Example Commands for Windows
# Create the example document
$ curl --anyauth --user user:password -X PUT -d @./example.json \
    -i -H "Content-type: application/json" \
    http://localhost:8003/v1/documents?uri=/docs/example.json

# Apply the patch
$ curl --anyauth --user user:password -X POST -d @./patch.json \
    -i -H "Content-type: application/json" \
    -H "X-HTTP-Method-Override: PATCH" \
    http://localhost:8003/v1/documents?uri=/docs/example.json

# Retrieve the results
curl --anyauth --user user:password -X GET \
```

```
-i -H "Accept: application/json" \
http://localhost:8003/v1/documents?uri=/docs/example.json
```

3.4.7.2 Example: Insert

The following patch inserts a new key-value pair to the root object in the first child position, inserts a new array element value in the `child3` subject object, and inserts a new key-value pair in the root object in first child position.

```
{ "patch": [
    { "insert": {
          "context": "$.parent.child1",
          "position": "before",
          "content": { "INSERT1": "INSERTED1" }
    }},
    { "insert": {
          "context": "$.parent.child3[1]",
          "position": "after",
          "content": "INSERT2"
    }},
    { "insert": {
          "context": "$.parent",
          "position": "last-child",
          "content": { "INSERT3": "INSERTED3" }
    }}
] }
```

The following table shows how applying the patch changes the target document.

Before Update	After Update
`{ "parent": {` `  "child1": {` `  "grandchild": "value"` `  },` `  "child2": "simple",` `  "child3": [ "av1", "av2" ],` `  "child4": [` `  { "a1" : "v1"},` `  { "a2": "v2" }` `  ]` `} }`	`{ "parent":{` `  **"INSERT1": "INSERTED1"**, ` "child1": {` ` "grandchild":"value"` ` },` ` "child2": "simple",` ` "child3": ["av1","av2",**"INSERT2"**],` ` "child4": [` ` {"a1":"v1"},` ` {"a2":"v2"}` `],` ` **"INSERT3":"INSERTED3"**` `} }`

3.4.7.3 Example: Replace

The following patch replaces one key-value pair with another, replaces the simple value of a key-value pair, and replaces the array value of a key-value pair.

```
{ "patch": [
    { "replace": {
        "select": "$.parent.child1",
        "content": { "REPLACE1": "REPLACED1" }
    }},
    { "replace": {
        "select": "$.parent.child2",
        "content": "REPLACED2"
    }},
    { "replace": {
        "select": "$.parent.child3",
        "content": { "child3": [ "REPLACED3a", "REPLACED3b" ] }
    }}
] }
```

The following table shows how applying the patch changes the target document.

Before Update	After Update

`{ "parent": {`
`  "child1": {`
`  "grandchild": "value"`
`  },`
`  "child2": "simple",`
`  "child3": [ "av1", "av2" ]`
`} }`
	`{ "parent": {`
	`  **"REPLACE1": "REPLACED1"**,`
	`  "child2": **"REPLACED2"**,`
	`  "child3": **[**`
	`  **"REPLACED3a",**`
	`  **"REPLACED3b"**`
	`  **]**`
	`} }`

3.4.7.4 Example: Replace-Insert

The following patch adds a new key-value pair to the root object and inserts a new array element in the child subject object.

```
{ "patch": [
    { "replace-insert": {
        "select": "$.parent.['child0']",
        "context": "$.parent.['child1']",
        "position": "before",
        "content": { "child0": "INSERTED1" }
    }},
    { "replace-insert": {
        "select": "['child2']",
        "context": "$.parent",
        "position": "last-child",
```

```
            "content": { "REPLACE2" : "REPLACED2" }
        }},
        { "replace-insert": {
            "select": "$.parent.['child3'][*][?(@ = 'av2')]",
            "context": "$.parent.['child3']",
            "position": "last-child",
            "content": "REPLACED3"
        }},
        { "replace-insert": {
            "select": "@.a3",
            "context": "$.parent.['child4']",
            "position" : "last-child",
            "content": { "a3": "INSERTED4" }
        }}
    ] }
```

Notice that `select` can be either absolute:

```
"select": "$.parent.['child0']"
```

Or relative to `context`:

```
"select": "['child2']",
"context": "$.parent",
```

Also notice that you can use predicates in the JSONPath expressions. For example, the following path selects all children of `$.parent.child3` with the value `av2`:

```
"select": "$.parent.['child3'][*][?(@ = 'av2')]"
```

The `[*]` path step selects all children. The `[?(@ = 'av2']` path step filters (`?`) all occurrences of the current node (`@`) equal to '`av2`'.

The following table shows how applying the patch changes the target document.

Before Update	After Update

{ "parent": {
  "child1": {
  "grandchild": "value"
  },
  "child2": "simple",
  "child3": ["av1", "av2"],
  "child4": [
  { "a1" : "v1"},
  { "a2": "v2" }
  ]
} }
	{ "parent": {
	  "child0": "INSERTED1",
	  "child1": {
	  "grandchild": "value"
	  },
	  "REPLACE2": "REPLACED2",
	  "child3": ["av1", "REPLACED3"],
	  "child4": [
	  { "a1": "v1" },
	  { "a2": "v2" },
	  { "a3": "INSERTED4" }
	  ]
	} }

3.4.7.5 Example: Delete

The following patch removes key-value pairs and array elements at various levels of a document. See the table below for how the patch affects the example content.

```
{ "patch": [
    { "delete" : {
        "select": "$..['DELETE1']"
    }},
    { "delete" : {
        "select": "$.parent.child2[?(@ = 'DELETE2')]"
    }},
    { "delete" : {
        "select": "$.parent['DELETE3']"
    }},
    { "delete" : {
        "select": "$.parent.child3[*][?(@ = 'DELETE4')]"
    }},
    { "delete" : {
        "select": "$..[*][?(@['DELETE5'])]"
    }}
] }
```

The following table shows how applying the patch changes the target document.

Before Update	After Update

{ "parent": {
"child1": {
"DELETE1": "value"
},
"child2": "DELETE2",
"DELETE3": "value",
"child3": ["av1", "DELETE4"],
"child4": [
{ "DELETE5" : "v1"},
{ "a2": "v2" }
]
} }
	{ "parent": {
	"child1": { },
	"child3": ["av1"],
	"child4": [
	{ "a2": "v2" }
	]
	} }

Whether or not a null object or array remains behind after deletion depends upon what is selected by the select JSONPath expression. For example, consider the delete operation that gets applied to `$parent.child1`:

```
{ "delete" : {
    "select": "$..['DELETE1']"
}}
```

This path recursively matches all occurrences of key `DELETE1` under the root node. Since it matches against a key, the operation deletes the matching key-value pair, but leaves an empty object behind:

```
"child1": { }
```

By contrast, the delete operation that applies to `child4` matches an object:

```
{ "delete" : {
    "select": "$..[*][?(@['DELETE5'])]"
}}
```

This path recursively matches all subobjects (`$..[*]`) and then filters out any objects that don't have a key of `DELETE5`. Since the expression matches an object, the delete removes the entire object, resulting in the following:

```
"child4": [
  { "a2": "v2" }
]
```

The same behavior applies to arrays and array elements.

3.4.8 Patching Metadata

You can partially update metadata for any document type, including XML, JSON, text, and binary. To update a portion of the metadata for a document, send a POST request to the `/documents` service with a URI of the following form.

```
http://host:port/version/documents?uri=document-uri&category=category
```

The category parameter should either be `metadata` or one of the metadata sub-categories of `collections`, `properties`, `permissions`, and `quality`. The category affects what pieces of metadata MarkLogic Server considers for updating. For example, if you use `category=permissions`, but your patch only contains an operation applied to properties, the patch has no effect.

Category does not affect how you construct path expressions in a patch. That is, `context` and `select` paths within the patch are not relative to the specified metadata category.

The request body must contain an XML or JSON patch descriptor. Construct your metadata patch to apply to the structure described in "Working with Metadata" on page 124, using the same syntax and semantics as for a content patch. For details, see in "XML Patch Reference" on page 64 or "JSON Patch Reference" on page 82.

> **Note:** Document quality (the `<quality/>` element or `quality` key-value pair) can only be the target of a replace operation.

You can update content and metadata in the same request by including operations on both in patch descriptor and setting the category request parameter appropriately. For example:

```
http://localhost:8003/v1/documents?uri=/my.xml&category=content&catego
ry=metadata
```

When patching content and metadata in the same request, the usual content type restrictions apply: You can only apply content patches to XML and JSON documents, and you cannot use a JSON patch descriptor on XML content, or vice versa.

Since JSON does not use qualified names, you can construct a `context` or `select` path that addresses both metadata and content items. If your content contains keys with the same name as the metadata keys (`properties`, `collections`, `permissions`, and `quality`) and you cannot construct an unambiguous path, do not combine content and metadata partial updates in the same request.

Paths for `context` and `select` can be relative or absolute with respect to the metadata structure. For example, both of the following operations insert a document into a new collection, using different `context` paths.

XML	JSON
<rapi:insert **context="/rapi:metadata/rapi:collections"** position="last-child"> <rapi:collection>INSERTED</rapi:collection> </rapi:insert>	{"insert": { **"context":"$.collections"**, "position":"last-child", "content":"INSERTED" }}
<rapi:insert **context="rapi:collections"** position="last-child"> <rapi:collection>INSERTED</rapi:collection> </rapi:insert>	{"insert": { **"context":"collections"**, "position":"last-child", "content":"INSERTED" }}

The following example is a patch that performs multiple metadata updates: Add the target document to a new collection, add or replace a property, and replace the document quality:

Format	Patch
XML	<rapi:patch xmlns:rapi="http://marklogic.com/rest-api" xmlns:prop="http://marklogic.com/xdmp/property"> <rapi:insert context="/rapi:metadata/rapi:collections" position="last-child"> <rapi:collection>INSERTED</rapi:collection> </rapi:insert> <rapi:replace-insert select="my-property" context="/rapi:metadata/prop:properties" position="last-child"> <my-property>REPLACE_OR_INSERT</my-property> </rapi:replace-insert> <rapi:replace select="/rapi:metadata/rapi:quality">2</rapi:replace> </rapi:patch>

Format	Patch

JSON	`{ "patch" : [`
	`  { "insert": {`
	`  "context": "$.collections",`
	`  "position": "last-child",`
	`  "content" : "INSERTED"`
	`  }},`
	`  { "replace-insert": {`
	`  "select": "my-property",`
	`  "context": "$.properties",`
	`  "position": "last-child",`
	`  "content" : { "my-property": "REPLACE_OR_INSERT" }`
	`  }},`
	`  { "replace": {`
	`  "select": "$.quality",`
	`  "content" : 2`
	`  }}`
	`] }`

Note: When patching properties using XML, you must explicitly declare the namespace `http://marklogic.com/xdmp/property` on the root `<patch/>` element because the `<properties/>` metadata element is in a different namespace from the other metadata elements.

3.4.9 Constructing Replacement Data on the Server

You can use builtin or user-defined replacement functions to generate the content for a partial update operation dynamically on MarkLogic Server. The builtin functions support simple arithmetic and string manipulation. For example, you can use a builtin function to increment the current value of numeric data or concatenate strings. For more complex operations, create and install a user-defined function.

The following topics are covered:

- Using a Replacement Constructor Function
- Using Builtin Replacement Constructors
- Writing a User-Defined Replacement Constructor
- Installing or Updating a User-Defined Replace Library

3.4.9.1 Using a Replacement Constructor Function

You can only use replacement generator functions with the `replace` and `replace-insert` operations. The MarkLogic REST API ships with a set of builtin arithmetic and string manipulation functions. If your patch includes a `replace-library` specification, you can also create user-defined functions.

The following example increases every `price` in an `inventory` element or object by 10% by using the builtin `ml.multiply` function to multiply the current value by 1.1.

XML	JSON
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api"> <rapi:replace select="/inventory/price" **apply="ml.multiply">1.1**</rapi:replace> </rapi:patch>	{"patch": [{"replace": { "select": "$.inventory.price", **"apply": "ml.multiply"**, "content": **1.1** } }] }

The following example uses a user-defined function to double the `price` in every `inventory` element or object. The `dbl` function is implemented by the XQuery module `/my.domain/my-lib.xqy`, which uses the module namespace `http://my/ns`. The `dbl` function does not expect any argument values, so there is no content included in the `replace` operation.

XML	JSON
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api"> **<rapi:replace-library** **at="/my.domain/my-lib.xqy"** **ns="http://my/ns" />** <rapi:replace select="/inventory/price" **apply="dbl"/**> </rapi:patch>	{"patch": [{**"replace-library": {** **"at": "/my.domain/my-lib.xqy",** **"ns": "http://my/ns"** } }, {"replace": { "select": "$.inventory.price", **"apply": "dbl"** } }] }

Use the following guidelines to specify a replacement generator function in a patch descriptor:

1. Set the value of the `apply` XML attribute or JSON key-value pair to the function localname.

2. If the function is a user-defined function, include a `replace-library` XML element or JSON key-value pair in the `patch` to specify the XQuery module containing the function implementation. The implementation module must be installed in the modules database of the REST API instance.

3. If the function expects arguments, specify them as child nodes of the operation element in XML or in the value of the `content` key-value pair in JSON. See the example below.

If a function expects a single argument, you can place the value directly in the content or in a `<rapi:value/>` XML element or `$value` JSON key-value pair. The previous example places the value (1.1) directly in the content section of the patch.

If a function expects multiple arguments, enclose each value in a `<rapi:value/>` XML element or `$value` JSON key-value pair. For example, the following patch uses the builtin function `ml.concat-between` which expects two string parameters.

XML	JSON
<rapi:patch xmlns:rapi="http://marklogic.com/rest-api"> <rapi:replace select="/my/data" apply="ml.concat-between"> **<rapi:value>my prefix</rapi:value>** **<rapi:value>my suffix</rapi:value>** </rapi:replace> </rapi:patch>	{"patch": [{"replace": { "select": "$.my.data", "apply": "ml.concat-between", **"content": [** **{ "$value" : "my prefix" },** **{ "$value" : "my suffix" }**] } }] }

In XML, the implicit datatype of arguments is xs:`untypedAtomic`, though a generator function can cast the value to the expected parameter type. For example, the builtin arithmetic functions cast to `xs:double` when possible. Use `@xsi:type` to explicitly specify a type on a value.

The following example specifies an explicit `xs:date` datatype. Note that you must declare the xs and `xsi` namespace aliases on the root of your `patch`:

```
<rapi:patch
    xmlns:rapi="http://marklogic.com/rest-api"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <rapi:replace select="/elem" apply="my-func">
    <rapi:value xsi:type="xs:date">2013-03-15</rapi:value>
  </rapi:replace>
</rapi:patch>
```

In JSON, use `$datatype` to explicitly specify an `xsi:type` for a `$value`. Omit the `xs:` namespace prefix on the type name. The following example explicitly specifies `xs:date` as the datatype for the replace function input parameter:

```
{"patch": [
  {"replace": {
    "select": "$.my.data",
    "apply": "ml.concat-between",
    "content": [
      { "$value" : "2013-03-15", $datatype: "date"}
    ]
```

```
    } }
  ] }
```

3.4.9.2 Using Builtin Replacement Constructors

The MarkLogic REST API includes several builtin server-side functions you can use to dynamically generate the content for a `replace` or `replace-insert` operation. For example, you can use a builtin function to increment the current value of a data item.

All the builtin function names have a "`ml.`" prefix. You cannot use this prefix on user-defined replacement generator functions.

The builtin arithmetic functions are equivalent to the XQuery `+`, `-`, `*`, and `div` operators, and accept values castable to the same datatypes. That is, numeric, date, dateTime, duration, and Gregorian (`xs:gMonth`, `xs:gYearMonth`, etc.) values. The operand type combinations are as supported by XQuery; for details, see http://www.w3.org/TR/xquery/#mapping. All other functions expect values castable to string.

The table below lists the builtin replacement generator functions provided by the MarkLogic REST API. In the table, *`$current`* represents the current value of the target of the replace operation; *`$arg`* and *`$argN`* represent argument values passed in by the patch.

Function Name	Num Args	Effect
`ml.add`	1	`$current + $arg`
`ml.subtract`	1	`$current - $arg`
`ml.multiply`	1	`$current * $arg`
`ml.divide`	1	`$current div $arg`
`ml.concat-before`	1	`fn:concat($arg, $current)`
`ml.concat-after`	1	`fn:concat($current, $arg)`
`ml.concat-between`	2	`fn:concat($arg1, $current, $arg2)`
`ml.substring-before`	1	`fn:substring-before($current, $arg)`
`ml.substring-after`	1	`fn:substring-after($current, $arg)`
`ml.replace-regex`	2 or 3	`fn:replace($current, $arg1, $arg2, $arg3)`

3.4.9.3 Writing a User-Defined Replacement Constructor

You can create your own XQuery functions to generate content for the `replace` and `replace-insert` operations. To make your functions available to partial update operations, install it in the modules database associated with your REST API instance; for details, see "Installing or Updating a User-Defined Replace Library" on page 105.

A user-defined replacement generator function has the following interface:

```
declare function module-ns:func-name(
  $current as node()?,
  $args as item()*
) as node()*
```

The current node (`$current`) is empty only when the function is invoked as an insert on behalf of a `replace-insert`.

The argument list supplied by the operation is passed through `$args`. You are responsible for validating the argument values. If the content supplied by the patch operation is an array of objects with `$value` key in JSON or a sequence of `rapi:value` elements in XML, the argument list is produced by applying `fn:data` function to each value. If an explicit datatype is specified by the patch operation, the cast is applied before invoking your function.

For JSON content, your function must be aware of how JSON documents are internalized as XML by MarkLogic Server. When your function is invoked on behalf of a JSON document inserted into the database using the MarkLogic REST API, `$current` represents the current node in the XML representation of the original JSON input. For details, see `json:transform-to-json` and Working With JSON in *Application Developer's Guide*.

Your function should report errors using `fn:error` and `RESTAPI-EXTNERR`. For details, see "Reporting Errors" on page 274.

The following example XQuery library module implements two replacement content constructors, `mylib:dbl` to double the value in a target node, and `my-lib:min` to replace the value of a node with the minimum value of the current node and a sequence of values passed in by the patch. For simplicity, this example skips most of the input data validation that a production implementation should include.

```
xquery version "1.0-ml";

module namespace my-lib =
  "http://marklogic.com/example/my-lib";

(: Double the value of a node :)
declare function my-lib:dbl(
  $current as node()?,
  $args as item()*
) as node()*
{
```

```
  if ($current/data() castable as xs:decimal)
  then
    let $new-value := xs:decimal($current/data()) * 2
    return element {fn:node-name($current)} {$new-value}
  else
    fn:error((), "RESTAPI-EXTNERR",
      ("400", "Bad Request", "xml", <reason>Non-decimal data</reason>))
};

(: Find the minimum value in a sequence of value composed of :)
(: the current node and a set of input values. :)
declare function my-lib:min(
  $current as node()?,
  $args as item()*
) as node()*
{
  let $new-value := fn:min(($current/data(), $args))
  return element {fn:node-name($current)} {$new-value}
};
```

The following patch snippet uses the above module, assuming the module is installed in the modules database with the URI `/ext/replace/my-lib.xqy`:

Format	Patch

XML	`<rapi:patch`
	`  xmlns:rapi="http://marklogic.com/rest-api">`
	`  **<rapi:replace-library at="/ext/replace/my-lib.xqy"**`
	`  **ns="http://marklogic.com/example/my-lib" />**`
	`  <rapi:replace select="/parent/child" apply="dbl"/>`
	`</rapi:patch>`

JSON	`{"patch": [`
	`  {**"replace-library": {**`
	`  **"at": "/ext/replace/my-lib.xqy",**`
	`  **"ns": "http://marklogic.com/example/my-lib"**`
	`  } },`
	`  {"replace": {`
	`  "select": "$.parent.child",`
	`  "apply": "dbl"`
	`  } }`
	`] }`

3.4.9.4 Installing or Updating a User-Defined Replace Library

To install user-defined replacement constructor functions, place your function(s) into an XQuery library module and install the module in the modules database associated with your REST API instance.

Your implementation, including any dependent libraries, must be installed in the modules database associated with your REST API instance. The simplest way to achieve this is to use the `/ext` service that is part of the MarkLogic REST API. For details, see "Managing Dependent Libraries and Other Assets" on page 299.

To install your library using the /ext service, send a PUT request to a URL of the following form, with your library module implementation in the request body:

```
http://host:port/version/ext/your/path/your-lib.xqy?perm:perm
```

If you do not specify any permissions, the module will only be executable by users with the `rest-admin` role.

When you use `/ext` to install your module, MarkLogic Server installs the XQuery library module in the request body into the modules database associated with your REST API instance, at a URI derived from the above URL, beginning with `/ext`. For example, the following command installs the library module with the URI `/ext/replace/my-lib.xqy`:

```
$ curl --anyauth --user user:password -X PUT -d @./my-lib.xqy \
    -i -H "Content-type: application/xquery" \
  http://localhost:8003/v1/ext/replace/my-lib.xqy?perm:my-user:execute
```

The following patch snippet uses the module URI in the value of the at XML attribute or JSON key in a replace-lib directive:

```
<rapi:replace-library at="/ext/replace/my-lib.xqy"
   ns="http://marklogic.com/example/my-lib" />
```

If your library module requires dependent libraries, you can install them in the same database directory (`/ext/replace/`, in the above example). The `/ext` service allows you to manage at both the directory and file level. For details, see "Managing Dependent Libraries and Other Assets" on page 299.

Use the same procedure to update your implementation after initial installation.

3.4.10 How Position Affects the Insertion Point

The `insert` and `replace-insert` patch operations include a position member that defines the point of insertion when coupled with a context expression. This section describes the details of how position affects the insertion point. The following topics are covered:

- Specifying Position in XML
- Specifying Position in JSON

3.4.10.1 Specifying Position in XML

The `@position` of an `<insert/>` or `<replace-insert/>` operation specifies where to insert new content, relative to the `context` node or attribute. Position can be one of the following values:

- `before`: Insert before the element or attribute identified by `@context`. The `@context` must not target the root element.
- `after`: Insert after the element or attribute identified by `@context`. The `@context` must not target the root element.
- `last-child`: Insert as the last child of the element identified by `@context`. The `@context` must target a node, not an attribute.

The `last-child` is not meaningful for operations on attributes.

The following table shows example combinations of `@context` and `@position` and the resulting insertion point for new content. The insertion point is indicated by `***`.

Context	Position	Example Insertion Point

`/parent/child[1]`	`before`	`<parent>`
		`  ***`
		`  <child>`
		`  <grandchild/>`
		`  </child>`
		`  <child a1="val" a2="val"/>`
		`</parent>`

`/parent/child`	`after`	`<parent>`
		`  <child>`
		`  <grandchild/>`
		`  </child>`
		`  ***`
		`  <child a1="val" a2="val"/>`
		`  ***`
		`</parent>`

`/parent/child[1]`	`last-child`	`<parent>`
		`  <child>`
		`  <grandchild/>`
		`  ***`
		`  </child>`
		`  <child a1="val" a2="val"/>`
		`</parent>`

`/parent/child/@a1`	`before`	`<parent>`
		`  <child>`
		`  <grandchild/>`
		`  </child>`
		`  <child *** a1="val" a2="val"/>`
		`</parent>`

Context	Position	Example Insertion Point

`/parent/child/@a1`	`after`	`<parent>`
		`  <child>`
		`  <grandchild/>`
		`  </child>`
		`  <child a1="val" *** a2="val"/>`
		`</parent>`

`/parent/child`	`last-child`	`<parent>`
		`  <child ***>`
		`  <grandchild/>`
		`  </child>`
		`  <child a1="val" a2="val" *** />`
		`</parent>`

`/parent/child/@a1`	`last-child`	Error. You cannot insert into an attribute.

3.4.10.2 Specifying Position in JSON

The `position` key-value pair in a patch specifies where to insert new content relative to the `context` item. Position can be one of the following values:

- `before`: Insert before the key-value pair or array element value selected by `context`.
- `after`: Insert after the key-value pair or array element value selected by `context`.
- `last-child`: Insert into the value of the target key-value pair or into target array, in the position of last child. The value selected by `context` must be an object or array.

The same usage applies whether inserting on behalf of an `insert` operation or a `replace-insert` operation.

The following table shows example combinations of `context` and `position` and the resulting insertion point for new content. The insertion point is indicated by `***`.

Context	Position	Example Insertion Point

`$.parent.child1`	`after`	`{"parent" : {`
		`  "child1" : [ "val1", "val2" ],`
		`  ***`
		`  "child2" : [`
		`  {"one": "val1"},`
		`  {"two": "val2"}`
		`  ]`
		`} }`

`$.parent.child1[0]`	`before`	`{"parent" : {`
		`  "child1" : [ *** "val1", "val2" ],`
		`  "child2" : [`
		`  {"one": "val1"},`
		`  {"two": "val2"}`
		`  ]`
		`} }`

`$.parent.child1`
an array
	`last-child`	`{"parent" : {`
		`  "child1" : [ "val1", "val2" ***],`
		`  "child2" : [`
		`  {"one": "val1"},`
		`  {"two": "val2"}`
		`  ]`
		`} }`

`$.parent`
an object
	`last-child`	`{"parent" : {`
		`  "child1" : [ "val1", "val2"],`
		`  "child2" : [`
		`  {"one": "val1"},`
		`  {"two": "val2"}`
		`  ]`
		`  ***`
		`} }`

`$.parent.child1[0]` a string	`last-child`	Error because the selected value is not an object or array.

3.4.11 Path Expressions Usable in Index Definitions

A patch operation uses one or more path expressions to identify the target of the operation.

For performance reasons, the path expression used in the `context` and `select` specifications of a patch operation is restricted to an XPath expression that can be used to define a path range index. For details, see Understanding Path Range Indexes in *Administrator's Guide*.

JSONPath expressions are constrained by equivalent limitations. For example, the only operators you can use in predicate expressions are comparison operators.

3.4.12 Introduction to JSONPath

When applying partial updates to JSON documents, you specify the target of a patch operation using JSONPath expressions in the context and select key-value pairs of a patch operation. This section gives a brief overview of JSONPath syntax. For a complete description of JSONPath, see http://goessner.net/articles/JsonPath/.

- The anonymous root of a JSON object is identified by a dollar sign ($).
- Delimit path steps by either periods (.) or square brackets ([]).
- Use square brackets to enclose predicate expressions and array element selectors.
- The first element in an array has position 0, which is different from XPath.
- Predicates must be enclosed in ?(), as in `[?(@.value < 10)]`.

The set of JSONPath expressions usable for document operations like partial update is limited to JSONPath equivalent to XPath expressions that can be used to define a path range index. For details, see “Path Expressions Usable in Index Definitions” on page 109.

The following table contains some simple JSONPath examples that satisfy the restrictions. The bold text indicates what is selected by each expression.

JSONPath	Selection

`$.parent`
 	
`$['parent']`
	{ **"parent": [**
	{"child": ["one", "two"] },
	{"child": {"grandchild": "three" }}
	] }

`$.parent[0]`
 	
`$['parent'][0]`
	{ "parent": [
	{"child": ["one", "two"] },
	{"child": {"grandchild": "three" }}
	] }

`$.parent['child']`	{ "parent": [**{"child": ["one", "two"] },** **{"child": {"grandchild": "three" }**}] }
`$.parent.child[*]['one']`	{ "parent": [{"child": [**"one"**, "two"] }, {"child": {"grandchild": "three" }}] }

3.5 Performing a Lightweight Document Check

Use this method to:

- Test for the existence of a document in the database.
- Retrieve a document identifier without fetching content or metadata when content versioning is enabled.
- Determining the total length of a document for setting the end boundary when iterating over content ranges in binary documents.

To perform a document check, construct a HEAD request to a URL of the form:

```
http://host:port/version/documents?uri=document_uri
```

The following example sends a HEAD request for an XML document:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X HEAD \
    -H "Accept: application/xml" \
    http://localhost:8004/v1/documents?uri=/xml/box.xml
...
Content-type: application/xml
Server: MarkLogic
Connection: close

HTTP/1.1 200 Document Retrieved
vnd.marklogic.document-format: xml
Content-type: application/xml
Server: MarkLogic
Connection: close
```

Issuing the same request on a non-existent document returns status 404:

```
$ curl --anyauth --user user:password -X HEAD \
    -H "Accept: application/xml" \
    http://localhost:8004/v1/documents?uri=/xml/dne.xml
...
Content-type: application/xml
Server: MarkLogic
Connection: close

HTTP/1.1 404 Not Found
Content-type: application/xml
Server: MarkLogic
Connection: close
```

3.6 Removing Documents from the Database

This section covers using the `/documents` and `/search` services to remove documents from the database. The following topics are covered:

- Removing a Document or Metadata
- Removing Multiple Documents
- Removing All Documents
- Removing a Semantic Graph

3.6.1 Removing a Document or Metadata

To remove a document and its metadata from the database, construct a DELETE request with a URL of the form:

```
http://host:port/version/documents?uri=document_uri
```

When you delete a document, its metadata is also deleted.

To remove or reset just metadata for a document, construct a DELETE request with a URL of the form:

```
http://host:port/version/documents?uri=document_uri&category=metadata_
category
```

Where `category` can appear multiple times, with the values described in “Metadata Categories” on page 124.

Deleting permissions resets the document permissions to the default permissions for the current user. Deleting quality resets the document quality to the default (0).

Deleting a binary document with extracted metadata stored in a separate XHTML document also deletes the XHTML metadata document. For more information, see “Working with Binary Documents” on page 123.

3.6.2 Removing Multiple Documents

You can remove all documents in a collection by sending a DELETE request with a URL of the following form:

```
http://host:port/version/search?collection=collection_name
```

Similarly, you can remove all documents in a directory by sending a DELETE request with a URL of the following form:

```
http://host:port/version/search?directory=directory_name
```

Where *directory_name* is the name of a directory in the database. The directory name must include a trailing “/”.

You can specify only one collection or one directory in a single request.

Note: Failing to specify either a directory or a collection removes all documents in the database.

Removing a documents also removes its metadata. Deleting a binary document with extracted metadata stored in a separate XHTML document also deletes the XHTML metadata document. For more information, see “Working with Binary Documents” on page 123.

The following example remove all documents in the “/plays” directory:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -i -X DELETE \
    http://localhost:8003/v1/search?directory=/plays/
...
HTTP/1.1 204 Updated
Server: MarkLogic
Content-Length: 0
Connection: close
```

3.6.3 Removing All Documents

To remove all documents in the database, send a DELETE request with a URL of the following form:

```
http://host:port/version/search
```

Clearing the database requires the `rest-admin` role or equivalent.

There is no confirmation or other safety net when you clear the database in this way. Creating a backup is advised.

The following example removes all documents and metadata from the content database:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -i -X DELETE \
    http://localhost:8003/v1/search
```

3.6.4 Removing a Semantic Graph

To remove the triples in a graph, send a DELETE to the `/graphs` service with a URL of one of the following forms:

```
http://host:port/version/graphs?graph=graph-uri

http://host:port/version/graphs?default
```

The `/graphs` service only affects triples stored in a triple document (a document with a `<sem:triples>` root element). Therefore, if the graph includes triples embedded in normal XML documents, the embedded triples are unaffected and the graph will continue to exist.

You can also delete a graph by using an empty graph in a PUT request to the `/graphs` service.

For details, see *Semantics Developer's Guide*.

3.7 Using Optimistic Locking to Update Documents

An application using optimistic locking creates a document only when the document does not exist and updates or deletes a document only when the document has not changed since this application last changed it. However, optimistic locking does not actually involve placing a lock on document.

Optimistic locking is useful in environments where integrity is important, but contention is rare enough that it is useful to minimize server load by avoiding unnecessary multi-statement transactions.

This section covers the following topics:

- Understanding Optimistic Locking
- Enabling Optimistic Locking
- Obtaining a Version Id
- Applying a Conditional Update
- Example: Updating a Document Using Optimistic Locking

3.7.1 Understanding Optimistic Locking

Consider an application that reads a document, makes modifications, and then updates the document in the database with the changes. The traditional approach to ensuring document integrity is to perform the read, modification, and update in a multi-statement transaction. This holds a lock on the document from the point when the document is read until the update is committed. However, this pessimistic locking blocks access to the document and incurs more overhead on the App Server.

With *optimistic locking*, the application does not hold a lock on a document between read and update. Instead, the application saves the document state on read, and then checks for changes at the time of update. The update fails if the document has changed between read and update. This is a *conditional update*.

Optimistic locking is useful in environments where integrity is important, but contention is rare enough that it is useful to minimize server load by avoiding unnecessary multi-statement transactions. The MarkLogic REST API also supports multi-statement transactions. For details, see "Managing Transactions" on page 250.

The MarkLogic REST API uses content versioning to implement optimistic locking. When content versioning is enabled, MarkLogic Server associates an opaque version id with a document. The version id changes each time you update the document. The version id is returned when you read a document, and you can pass it back in an an update or delete operation to test for changes prior to commit.

> **Note:** Content versioning in the MarkLogic REST API does *not* implement document versioning. When content versioning is enabled, MarkLogic Server does not keep multiple versions of a document or track what changes occur. The version id can only be used to detect that a change occurred.

Using optimistic locking in your application requires the following steps:

1. Enabling Optimistic Locking by setting the `update-policy` REST instance configuration property.

2. Obtaining a Version Id for documents you wish to conditionally update.

3. Applying a Conditional Update by supplying the version id in your update operations.

3.7.2 Enabling Optimistic Locking

Enable optimistic locking using the `update-policy` instance configuration property, as described in “Configuring Instance Properties” on page 41. For example, the following command sets `update-policy` to `version-optional`:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT \
    -d'{"update-policy":"version-optional"}' \
    -H "Content-type: application/json" \
    http://localhost:8003/v1/config/properties
```

The `update-policy` property can be set to `merge-metadata` (the default), `version-required`, or `version-optional`. Set the property to `version-required` if you want every document update or delete operation to use optimistic locking. Set the property to `version-optional` to allow selective use of optimistic locking.

> **Note:** The `update-policy` property replaces the older `content-versions` policy; `content-versions` is deprecated. Setting `update-policy` to `version-optional` is equivalent to setting `content-versions` to `optional`. Setting `update-policy` to `version-required` is equivalent to setting `content-versions` to `required`.

The table below describes how each setting for this property affects document operations.

Setting	Effect
`merge-metadata`	This is the default setting. If you insert, update, or delete a document that does not exist, the operation succeeds. If a version id is present, it is ignored.
`version-optional`	If you insert, update or delete a document that does not exist, the operation succeeds. If a version id is present, the operation fails if the document exists and the current version id does not match the supplied version id.
`version-required`	If you update or delete a document without supplying a version id and the document does not exist, then the operation succeeds; if the document exists, the operation fails. If a version id is present, the operation fails if the document exists and the current version id does not match the version in the header.
`overwrite-metadata`	The behavior is the same as `merge-metadata`, except that metadata in the request overwrites any pre-existing metadata, rather than being merged into it. This setting disables optimistic locking.

3.7.3 Obtaining a Version Id

When optimistic locking is enabled, a version id is included in the response when you read documents. You can only obtain a version id if optimistic locking is enabled by setting `update-policy`; for details, see “Enabling Optimistic Locking” on page 115.

You can obtain a version id for use in conditional updates in the following ways:

- Perform a single document GET or HEAD request to `/documents`. The version id is returned in the ETag header of the response.
- Perform a multi-document read request, such as GET `/documents` with multiple URIs. The version id is returned in the Content-Disposition header of each content part in the multipart response.

You can mix and match these methods for obtaining a version id.

The following example command returns the version id for a single document:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -i -X HEAD \
    -H "Accept: application/xml" \
    http://localhost:8003/v1/documents?uri=/docs/example.xml
```

```
...
HTTP/1.1 200 Document Retrieved
Content-type: application/xml
ETag: "13473172834878540"
Server: MarkLogic
Connection: close
```

The following example command returns the version id for multiple documents in a single request:

```
curl --anyauth --user user:password -X GET -i \
  -H "Accept: multipart/mixed; boundary=BOUNDARY" \
  http://localhost:8003/v1/documents?uri=doc1.xml\&uri=doc2.json
...
--BOUNDARY
Content-Type: application/xml
Content-Disposition: attachment; filename="doc1.xml";
category=content; format=xml; versionId=14075140367230760
Content-Length: 87

...document contents...
--BOUNDARY
Content-Type: application/json
Content-Disposition: attachment; filename="doc2.json";
category=content; format=json; versionId=14075140367230760
Content-Length: 15

...document contents...
```

For more information on multi-document requests, see “Reading and Writing Multiple Documents” on page 206.

3.7.4 Applying a Conditional Update

To apply a conditional update, supply a version id on your update operation. The version id is ignored if optimistic locking is not enabled; for details, see “Enabling Optimistic Locking” on page 115.

When a document update includes a version id, MarkLogic Server checks for changes to the version id before committing the update (or delete) operation. If the version id has changed, the update fails. In a multi-document update, the entire batch of updates is rejected if any conditional update fails.

You can supply a version id on an update in the following ways:

- Pass the version id in the If-Match header of a single document PUT, POST, PATCH, or DELETE request to `/documents`.
- Pass the version id in the Content-Disposition header of a content part in a multi-document POST request to `/documents`.

The following example performs a conditional update of a single document by passing a version id in the If-Match request header:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -i -X PUT -d"<modified-data/>"  \
    -H "Content-type: application/xml" \
    -H "If-Match: 13473769780393030" \
    http://localhost:8003/v1/documents?uri=/docs/example.xml
```

The following POST body excerpt for a multi-document update unconditionally updates the first document (`doc1.xml`) and conditionally updates the second document (`doc2.xml`). Note that if `update-policy` is set to `version-required`, the request will fail if `doc1.xml` already exists because no version id is supplied for it.

```
--BOUNDARY
Content-Type: application/xml
Content-Disposition: attachment; filename="doc1.xml"
Content-Length: 87

...document contents...
--BOUNDARY
Content-Type: application/json
Content-Disposition: attachment; filename="doc2.json";
versionId=14075140367230760
Content-Length: 15

...document contents...
--BOUNDARY--
```

The following command applies the above update, if the complete POST body is in the file `/example/optimistic-locking`.

```
$ curl --anyauth --user user:password -X POST -i \
  --data-binary @/example/optimistic-locking \
  -H "Content-type: multipart/mixed; boundary=BOUNDARY" \
  -H "Accept: application/xml" \
  http://localhost:8003/v1/documents
```

For more information on multi-document requests, see “Reading and Writing Multiple Documents” on page 206.

3.7.5 Example: Updating a Document Using Optimistic Locking

The following example demonstrates using optimistic locking to conditionally update a single document. Both a successful and a failed update are shown.

This example uses a PUT request to demonstrate a conditional update. For a POST, PATCH, or DELETE operation, follow a similar procedure, passing the version id in the If-Match header of your update or delete request. For an equivalent multi-document update, pass the version id in the Content-Disposition header of each content part.

1. If content versioning is not already enabled, enable it by setting `update-policy` to `version-optional`:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT \
    -d'{"update-policy":"version-optional"}' \
    -H "Content-type: application/json" \
    http://localhost:8003/v1/config/properties
```

2. Insert an example document into the database to initialize the example:

```
$ curl --anyauth --user user:password -i -X PUT -d"<data/>" \
    -H "Content-type: application/xml" \
    http://localhost:8003/v1/documents?uri=/docs/example.xml
```

3. Unconditionally retrieve a local copy of the document. Note the version id is returned in the ETag header:

```
$ curl --anyauth --user user:password -i -X GET \
    -H "Accept: application/xml" \
    http://localhost:8003/v1/documents?uri=/docs/example.xml
...
HTTP/1.1 200 Document Retrieved
vnd.marklogic.document-format: xml
Content-type: application/xml
ETag: "13473769780393030"
Server: MarkLogic
Content-Length: 47
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<data/>
```

4. Conditionally upate the document by sending new contents with the version id from Step 3 in the If-Match header. Since the document has not changed since Step 3, the update succeeds. The document version id is changed by this operation.

```
$ curl --anyauth --user user:password -i -X PUT -d"<modified-data/>"  \
    -H "Content-type: application/xml" \
    -H "If-Match: 13473769780393030" \
    http://localhost:8003/v1/documents?uri=/docs/example.xml
...
HTTP/1.1 204 Content Updated
Server: MarkLogic
Content-Length: 0
Connection: close
```

5. To illustrate update failure when the version ids do not match, attempt to update the document again, using the version id from Step 3, which is no longer current. The update fails.

```
$ curl --anyauth --user user:password -i -X PUT -d"<data/>"  \
    -H "Content-type: application/xml" \
    -H "If-Match: 13473769780393030" \
    http://localhost:8003/v1/documents?uri=/docs/example.xml
...
HTTP/1.1 412 Precondition Failed
Content-type: application/xml
Server: MarkLogic
Content-Length: 370
Connection: close

<?xml version="1.0"?>
<rapi:error xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:status-code>412</rapi:status-code>
  <rapi:status>Precondition Failed</rapi:status>
  <rapi:message-code>RESTAPI-CONTENTWRONGVERSION</rapi:message-code>
  <rapi:message>RESTAPI-CONTENTWRONGVERSION: (err:FOER0000) Content
version mismatch:  uri: /docs/example.xml version:
13473788748796580</rapi:message>
</rapi:error>
```

3.8 Client-Side Cache Management Using Content Versioning

You can use content versioning to refresh a copy of a document stored on the client only if the document in the database has been modified. This section covers the following topics:

- Enabling Content Versioning
- Using Content Versioning for Cache Refreshing
- Example: Refreshing a Cached Document

3.8.1 Enabling Content Versioning

Enable content versioning using the `update-policy` instance configuration property, as described in “Configuring Instance Properties” on page 41.

The default update policy is `merge-metadata`. If you set `update-policy` to `version-required` or `version-optional`, content versioning is enabled, and a GET or HEAD request to `/documents` returns a version id in the ETag response header.

Note: Enabling content versioning can affects document insertion, update, and deletion. For details, see “Enabling Optimistic Locking” on page 115.

Note: Content versioning in the MarkLogic REST API does *not* implement document versioning. When content versioning is enabled, MarkLogic Server does not keep multiple versions of a document or track what changes occur. The version id can only be used to detect that a change occurred.

3.8.2 Using Content Versioning for Cache Refreshing

When content versioning is enabled, sending a GET request to `/documents` with a version id in the If-None-Match HTTP header only retrieves a new copy of the document if it has changed relative to the version id in the header.

> **Note:** The `update-policy` property replaces the older `content-versions` property; `content-versions` is deprecated. Setting `update-policy` to `version-optional` is equivalent to setting `content-versions` to `optional`. Setting `update-policy` to `version-required` is equivalent to setting `content-versions` to `required`.

Follow the procedure below to use content version for cache refreshing. For a complete example, see "Example: Refreshing a Cached Document" on page 121.

1. If content versioning is not already enabled, set the `update-policy` instance configuration property to `version-optional` or `version-required`; see "Enabling Content Versioning" on page 120.

2. Read a document by making a GET request to `/documents`. The response includes the version id in the ETag header.

3. Save the version id returned in the ETag response header.

4. When you want to refresh the cache, send another GET request to `/documents` with the version id from Step 2 in the If-None-Match HTTP header.

   If the current version id matches the one in the If-None-Match header, no document is retrieved and MarkLogic Server returns status 304. If the current version id differs from the one in the If-None-Match header, the document is returned, along with the new version id in the ETag response header.

3.8.3 Example: Refreshing a Cached Document

The following example demonstrates using content versioning to refresh a client-side document cache. The example includes includes a case where the document is unchanged in the database, as well as the case where the local cache is out of date.

1. If content versioning is not already enabled, enable it by setting `update-policy` to `version-optional`:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT \
    -d'{"update-policy":"version-optional"}' \
    -H "Content-type: application/json" \
    http://localhost:8003/v1/config/properties
```

2. Insert a sample document into the database to initialize the example:

```
$ curl --anyauth --user user:password -i -X PUT -d"<data/>" \
    -H "Content-type: application/xml" \
    http://localhost:8003/v1/documents?uri=/docs/example.xml
```

3. Unconditionally retrieve a local copy of the document, as if to cache it. Note the version id is returned in the ETag header:

```
$ curl --anyauth --user user:password -i -X GET \
    -H "Accept: application/xml" \
    http://localhost:8003/v1/documents?uri=/docs/example.xml
...
HTTP/1.1 200 Document Retrieved
vnd.marklogic.document-format: xml
Content-type: application/xml
ETag: "13473769780393030"
Server: MarkLogic
Content-Length: 47
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<data/>
```

4. Conditionally retrieve the document, as if to refresh the cache. Supply the version id from Step 3 in the If-None-Match header. Since the document has not changed, no content is retrieved.

```
$ curl --anyauth --user user:password -i -X GET \
    -H "Accept: application/xml" \
    -H "If-None-Match: 13473769780393030" \
    http://localhost:8003/v1/documents?uri=/docs/example.xml
...
HTTP/1.1 304 Content Version Not Modified
ETag: "13473769780393030"
Server: MarkLogic
Content-Length: 0
Connection: close
```

5. Update the document in the database, which changes the version id.

```
$ curl --anyauth --user user:password -i -X PUT -d"<modified-data/>"  \
    -H "Content-type: application/xml" \
    http://localhost:8003/v1/documents?uri=/docs/example.xml
...
HTTP/1.1 204 Content Updated
Server: MarkLogic
Content-Length: 0
Connection: close
```

6. Conditionally retrieve the document again, as if to refresh the cache. Supply the version id from Step 3 in the If-None-Match header. Since the document has changed, the content is retrieved. The new version id is also returned via the ETag header.

```
$ curl --anyauth --user user:password -i -X GET \
    -H "Accept: application/xml" \
    -H "If-None-Match: 13473769780393030" \
    http://localhost:8003/v1/documents?uri=/docs/example.xml
...
HTTP/1.1 200 Document Retrieved
vnd.marklogic.document-format: xml
Content-type: application/xml
ETag: "13473770707201670"
Server: MarkLogic
Content-Length: 56
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<modified-data/>
```

3.9 Working with Binary Documents

This section covers the following topics:

- Types of Binary Documents
- Streaming Binary Content

3.9.1 Types of Binary Documents

This section provides a brief summary of binary document types. For details, see Working With Binary Documents in the *Application Developer's Guide*.

MarkLogic Server can store binary documents in three representations:

- Small binary documents are stored entirely in the database.
- Large binary documents are stored on disk with a small reference fragment in the database. The on-disk content is managed by MarkLogic Server.
- External binary documents are stored on disk with a small reference fragment in the database. However, the on-disk content is not managed by MarkLogic Server.

Small and large binary documents are created automatically for you, depending on the document size. External binary documents cannot be created using the MarkLogic REST API.

Large binary documents can be streamed out of the database using Range requests. For details, see "Streaming Binary Content" on page 123.

3.9.2 Streaming Binary Content

Streaming binary content out of the database avoids loading the entire document into memory. You can stream binary documents by sending GET requests to `/documents` that include range requests under following conditions:

- The size of the binary content returned is over the large binary size threshold. For details, see Working With Binary Documents in the *Application Developer's Guide*.
- The request is for content only. That is, no metadata is requested.
- The MIME type of the content is determinable from the Accept header or the document URI file extension.

The following example requests the first 500K of the binary document with URI `/binaries/large.jpg`:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -i -o piece.jpg -X GET \
    -H "Accept: application/jpg" -r "0-511999" \
    http://localhost:8003/v1/documents?uri=/binaries/large.jpg
...
HTTP/1.1 206 Binary Part Retrieved
Content-type: application/jpeg
Content-Range: bytes 0-511999/533817
Server: MarkLogic
Content-Length: 511999
Connection: close
```

3.10 Working with Metadata

The `/documents` service supports inserting, updating, and retrieving document metadata. Metadata is the properties, collections, permissions, and quality of content.

Metadata manipulation is most often exposed by the MarkLogic REST API through a `category` URL parameter. Metadata can be passed around as either XML or JSON, usually controlled through either an HTTP header or a `format` URL parameter. For specifics, see the *MarkLogic REST API Reference*.

This section covers the following topics related to metadata storage and retrieval:

- Metadata Categories
- XML Metadata Format
- JSON Metadata Format
- Working With Properties Using JSON
- Disabling Metadata Merging

3.10.1 Metadata Categories

Where the MarkLogic REST API accepts specification of metadata categories, the following categories are recognized:

- `collections`

- `permissions`
- `properties`
- `quality`
- `metadata`

The `metadata` category is shorthand for all the other categories. That is, `metadata` includes collections, permissions, properties, and quality.

Some requests also support a `content` category as a convenience for requesting or updating both metadata and document content in a single request.

3.10.2 XML Metadata Format

Metadata contains information about document collections, permissions, properties, and quality. The format is fully described by the schema file:

```
MARKLOGIC_INSTALL_DIR/Config/restapi.xsd
```

The following is a summary of the structure of the metadata. All elements are in the namespace `http://marklogic.com/rest-api`. You can have 0 or more `<collection/>`, `<permission/>` or property elements. There can be only one `<quality/>` element. The element name and contents of each property element depends on the property.

```
<metadata xmlns="http://marklogic.com/rest-api">
  <collections>
    <collection>collection-name</collection>
  </collections>
  <permissions>
    <permission>
      <role-name>name</role-name>
      <capability>capability</capability>
    </permission>
  </permissions>
  <properties>
    <property-element/>
  </properties>
  <quality>integer</quality>
</metadata>
```

The following example shows a document in 2 collections, with one permission and 2 properties.

```
<rapi:metadata xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:collections>
    <rapi:collection>shapes</rapi:collection>
    <rapi:collection>squares</rapi:collection>
  </rapi:collections>
  <rapi:permissions>
    <rapi:permission>
      <rapi:role-name>hadoop-user-read</rapi:role-name>
```

```
      <rapi:capability>read</rapi:capability>
    </rapi:permission>
  </rapi:permissions>
  <prop:properties xmlns:prop="http://marklogic.com/xdmp/property">
    <myprop>this is my prop</myprop>
    <myotherprop>this is my other prop</myotherprop>
  </prop:properties>
  <rapi:quality>0</rapi:quality>
</rapi:metadata>
```

3.10.3 JSON Metadata Format

Metadata contains information about document collections, permissions, properties, and quality. A block of metadata can contain multiple collections, permissions and properties, but only 1 quality. The structure of each property depends on the property contents.

```
{
  "collections" : [ string ],
  "permissions" :
    {
      "role-name" : string,
      "capabilities" : [ string ]
    }
  ],
  "properties" : {
    property-name : property-value
  },
  "quality" : integer
}
```

The following example shows a document in 2 collections, with one permission and 2 properties.

```
{
  "collections": [
    "shapes",
    "squares"
  ],
  "permissions": [
    {
      "role-name": "hadoop-user-read",
      "capabilities": [
        "read"
      ]
    }
  ],
  "properties": {
    "myprop": "this is my prop",
    "myotherprop": "this is my other prop"
  },
  "quality": 0
}
```

3.10.4 Working With Properties Using JSON

In most cases, your application should work with properties in a consistent format. That is, if you insert or update properties as XML, then you should retrieve them in XML. If you insert or update properties as JSON, you should retrieve them in JSON. If you mix and match XML and JSON formats, you can encounter namespace inconsistencies.

Properties inserted as JSON are stored in the database as XML. When you insert a user-defined property using JSON, the XML representation is an XML element in the namespace `http://marklogic.com/xdmp/json/basic` with a localname that matches the property key name in JSON. For example, a property of the form `{ "my-property" : "value" }` in JSON has the following XML representation:

```
<rapi:metadata uri="/my/doc.json" ...>
  <prop:properties xmlns:prop="http://marklogic.com/xdmp/property">
    <my-property type="string"
        xmlns="http://marklogic.com/xdmp/json/basic">
      value
    </my-other-key>
  </prop:properties>
</rapi:metadata>
```

If you access it using XML, you must be aware of the namespace. Similarly, you can insert properties in no namespace or in your own namespace using XML, but the namespace cannot be reflected in the JSON representation of the property. Therefore, it is best to be consistent in how you work with properties.

The one exception is in code that works with properties on the server side, such as content and search result transformations. Such code always accesses properties as XML.

If you configure an index based on a user-defined property inserted using JSON, you should use the `http://marklogic.com/xdmp/json/basic` in your configuration. For details, see “Creating Indexes on JSON Keys” on page 150.

Protected system properties such as `last-updated` cannot be modified by your application. The JSON representation of such properties wraps them in an object with the key `$ml.prop`. For example:

```
{ "properties": {
    "$ml.prop": {
      "last-updated": "2013-11-06T10:01:11-08:00"
    }
} }
```

3.10.5 Disabling Metadata Merging

If you use the MarkLogic REST API to ingest a large number of documents at one time and you find the performance unacceptable, you might see a small performance improvement by disabling metadata merging. This topic explains the tradeoff and how to disable metadata merging.

- When to Consider Disabling Metadata Merging
- Understanding Metadata Merging
- How to Disable Metadata Merging

3.10.5.1 When to Consider Disabling Metadata Merging

The performance gain from disabling metadata merging is modest, so you are unlikely to see significant performance improvement from it unless you ingest a large number of documents. You might see a performance gain under one of the following circumstances:

- Ingesting a large number of documents, one at a time.
- Updating a large number of documents that share same metadata or that use the default metadata.

You cannot disable metadata merging in conjunction with update policies `version-optional` or `version-required`.

Metadata merging is disabled by default for multi-document write requests, as long as the request includes content for a given document. For details, see “Understanding When Metadata is Preserved or Replaced” on page 219.

3.10.5.2 Understanding Metadata Merging

Disabling metadata merging effectively eliminates the distinction between inserting a new document and updating an existing document, with respect to metadata handling.

Metadata merging is enabled by default. When you update metadata, the metadata in your update is merged with existing metadata. You can extend or update metadata without re-specifying unmodified properties, collections, permissions, or document quality. Similarly, when you update document content without including any metadata updates, the existing metadata is preserved.

When metadata merging is disabled, updating any metadata on a document overwrites all metadata for the document with the metadata provided in the request (plus default values for unspecified metadata categories). A content update that does not include any metadata resets the document metadata to the default values.

For example, if you set permissions on a document when you initially insert it into the database, and then later add the document to a collection while metadata merging is disabled, the permissions will be reset to default document permissions. In order to preserve the pre-existing permissions, you must specify them explicitly in your update.

3.10.5.3 How to Disable Metadata Merging

Metadata merging is controlled by the `update-policy` instance configuration property. The default value is `merge-metadata`.

To disable metadata merging, set `update-policy` to `overwrite-metadata`. using the procedure described in “Configuring Instance Properties” on page 41. For example:

```
$ cat props.xml
<properties xmlns="http://marklogic.com/rest-api">
  <update-policy>overwrite-metadata</update-policy>
</properties>

# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user admin:adm1n -X PUT -d@"./props.xml" \
    -H "Content-type: application/xml" \
    http://localhost:8003/v1/config/properties
```

The equivalent JSON input is shown below:

```
{
  "update-policy" : "overwrite-metadata"
}
```

4.0 Using and Configuring Query Features

This chapter covers the following topics:

- Query Feature Overview
- Querying Documents and Metadata
- Querying Lexicons and Range Indexes
- Using Query By Example to Prototype a Query
- Analyzing Lexicons and Range Indexes With Aggregate Functions
- Specifying Dynamic Query Options with Combined Query
- Querying Triples
- Configuring Query Options
- Using Namespace Bindings
- Generating Search Facets
- Paginating Results
- Customizing Search Results
- Generating Search Term Completion Suggestions

4.1 Query Feature Overview

The MarkLogic REST API includes several services for querying content in a MarkLogic Server database. The usage model for the query features is:

1. Optionally, use the `/config/query` service to install a set of named query options to apply to future queries. The name is used to apply the options to subsequent queries. For details, see “Configuring Query Options” on page 167.

2. Optionally, use the `/config/namespaces` service to install namespace bindings for namespace aliases you will use in queries. For details, see “Using Namespace Bindings” on page 177.

3. Use `/search`, `/keyvalue`, `/qbe`, or `/values` to perform a query.

   a. Depending on the service, the query may be expressed using a request parameter or in the request body. For details, see the reference documentation for each method.

   b. Apply options by specifying the name of previously installed query options in the `options` request parameter and/or providing dynamic query options in the request body using a combined query. For details, see “Adding Query Options to a Request” on page 168.

c. Specify the content type (XML or JSON) of the query results using the format request parameter or Accept headers.

4. Query results of the requested content type are returned in the response body.

By default, a query operation returns search response data that summarizes the matches and can contain snippets of matching content. This chapter focuses on such operations. However, you can also use `/search` and `/qbe` to retrieve the matching documents or metadata instead of a search response; for details, see "Reading and Writing Multiple Documents" on page 206.

The table below gives a brief summary of each query related service.

Service	Description
`/search`	Use string, structured, and combined queries to search documents and metadata. Queries may be expressed in XML or JSON. For details, see "Querying Documents and Metadata" on page 132.
`/keyvalue`	Use key-value pair queries to search XML content and metadata, and JSON content. For example, search by element, element attribute, or JSON key values. Dynamic query options are not supported by this service. For details, see "Querying Documents and Metadata" on page 132.
`/qbe`	Use Query By Example (QBE) for rapid prototyping of XML and JSON document searches. Queries using QBE syntax may be expressed in XML or JSON. For details, see "Using Query By Example to Prototype a Query" on page 152.
`/values`	Query lexicon and range index values and value co-occurrences; analyze lexicon and range index values and value-co-occurrences with builtin and user-defined aggregate functions. For details, see "Querying Lexicons and Range Indexes" on page 140.
`/config/query`	Use XML or JSON to configure persistent query options for use with `/search`, `/keyvalue`, and `/values`. For details, see "Configuring Query Options" on page 167.
`/config/namespaces`	Configure bindings between namespace prefixes and namespace URIs so you can use QNames in query contexts where it is not possible to dynamically specify a namespace. For details, see "Using Namespace Bindings" on page 177.

The `/search`, `/keyvalue`, and `/values` services use query options to control and configure queries and search results. For deatils, see:

- “Appendix A: JSON Query Options Reference” on page 308
- “Appendix B: XML Query Options Reference” on page 366

For more information on search concepts, see the Search Developer’s Guide.

4.2 Querying Documents and Metadata

This section describes how to use the `/search` and `/keyvalue` services to search documents and metadata.

- Constraining a Query by Collection or Directory
- Searching With String Queries
- Searching With Structured Queries
- Searching By JSON Key and Value
- Searching By Element or Attribute Name and Value
- Debugging /search Queries With Logging

You can use `/search` to retrieve a search response, matching documents and metadata, or both. The examples in this section only return a search response. To retrieve whole documents and/or their metadata, see “Reading Multiple Documents Matching a Query” on page 240.

4.2.1 Constraining a Query by Collection or Directory

Use the `collection` and `directory` request parameters to the `/search` and `/keyvalue` services to limit search results to matches in specific collections or database directories.

You can specify multiple collections or directories. For example, the following URL finds documents containing “julius” in the “tragedy” and “comedy” collections:

```
http://localhost:8003/v1/search?q=julius&collection=tragedy&collection
=comedy
```

You can use collection and directory constraints together. For example, by adding a directory to the above search, you can further limit matches to documents in the “/shakespeare/plays” directory:

```
http://localhost:8003/v1/search?q=julius&collection=tragedy&collection
=comedy&directory=/shakespeare/plays
```

For details about collections and directories, see Collections in the *Search Developer’s Guide* and Directories in the *Application Developer’s Guide*.

4.2.2 Searching With String Queries

The MarkLogic Server Search API default search grammar allows you to quickly construct simple searches such as “cat”, “cat AND dog”, or “cat NEAR dog”. You can also customize the search grammar. For details, see Search Grammar in the *Search Developer’s Guide*.

To search for matches to a simple string query, send a GET request to the `/search` service with a URL of the form:

```
http://host:port/version/search?q=query_string
```

Where *`query_string`* is a string conforming to the search grammar.

You can include both a string query and a structured query in the same request. In this case, the two queries are AND’d together. For more information, see “Searching With Structured Queries” on page 133 and “Specifying Dynamic Query Options with Combined Query” on page 161.

You can request search results in XML or JSON. Use the Accept header or `format` request parameter to select the output content type. For details, see “Controlling Input and Output Content Type” on page 25.

For details on the structure of search results, see “Appendix A: JSON Query Options Reference” on page 308 or “Appendix B: XML Query Options Reference” on page 366.

For a complete list of parameters available with the `/search` service, see “GET /v1/search” in the *REST Resources API*.

4.2.3 Searching With Structured Queries

Structured queries enable you to create complex queries, represented in XML or JSON. For details, see Searching Using Structured Queries in the *Search Developer’s Guide*.

To search using a structured query, send a GET or POST request to the `/search` service. To pass the structured query as a request parameter, send a GET request with a URL of the form:

```
http://host:port/version/search?structuredQuery=query
```

Where *`query`* is an XML or JSON representation of a structured query.

To pass a structured query in the request body, send a POST request of the following form and place a structured or combined query in the POST body. Set the Content-type header appropriately:

```
http://host:port/version/search
```

If the request also includes a string query, the string query and structured query are AND’d together. For details, see “Searching With String Queries” on page 133 and “Specifying Dynamic Query Options with Combined Query” on page 161.

You can request search results in XML or JSON. Use the Accept header or `format` request parameter to select the output content type. For details, see "Controlling Input and Output Content Type" on page 25.

For a complete list of parameters available with the `/search` service, see "GET /v1/search" or "POST /v1/search" in the *REST Resources API*.

The following example combines a structured query equivalent to "Yorick NEAR Horatio" and a string query for "knew", requesting search results in JSON. The effect is equivalent to searching for "(Yorick NEAR Horatio) AND knew".

```
$ cat sq.xml
<search:query xmlns:search="http://marklogic.com/appservices/search">
  <search:near-query>
    <search:term-query>
      <search:text>Yorick</search:text>
    </search:term-query>
    <search:term-query>
      <search:text>Horatio</search:text>
    </search:term-query>
  </search:near-query>
</search:query>

# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X POST -d@"./sq.xml" \
  -H 'Content-type: application/xml' \
  -H 'Accept: applicaton/json' \
  'http://localhost:8003/v1/search?q=knew'
```

4.2.4 Searching By JSON Key and Value

When you store JSON documents in the database using the `/documents` service of the MarkLogic REST API, you can search based on the values of keys in the JSON content using the `/keyvalue` service.

> **Note:** Efficient key-value searches require an element range index on the key. For details, see "Creating Indexes on JSON Keys" on page 150.

To search by key value, send a GET request to the `/keyvalue` service with a URL of the form:

```
http://host:port/version/keyvalue?key=the_key&value=the_value
```

Where `the_key` is the name of the key to match against `the_value`. For example, the following search matches all JSON documents containing a key called "species" with a value of "canis lupus":

```
http://localhost:2003/v1/keyvalue?key=species&value=canis%20lupus
```

Only JSON documents are considered in the search when you use the `key` parameter. To query XML content, see "Searching By Element or Attribute Name and Value" on page 136.

You can request search results in XML or JSON. Use the Accept header or `format` request parameter to select the output content type. For details, see "Controlling Input and Output Content Type" on page 25.

For a complete list of parameters available with the `/keyvalue` service, see "GET /v1/keyvalue" in the *REST Resources API*.

The following example searches for occurrences of the key "SPEAKER" with the value "BERNARDO". First, insert a JSON document into the database:

```
$ cat hamlet.json
{
  "TITLE":"Hamlet",
  "ACT":{
    "TITLE":"ACT I",
    "SCENE":{
      "TITLE":"SCENE I. Elsinore. A platform before the castle.",
      "STAGEDIR":"FRANCISCO at his post. Enter to him BENARDO",
      "SPEECHES":[
        {"SPEECH":{
          "SPEAKER":"BERNARDO",
          "LINE":"Whos there?"
        }},
        {"SPEECH":{
          "SPEAKER":"FRANCISCO",
          "LINE":"Nay, answerme: stand, and unfold yourself."
        }},
        {"SPEECH":{
          "SPEAKER":"BERNARDO",
          "LINE":"Long live the king!"
        }}
      ]
    }
  }
}

# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT -d @"./hamlet.json" \
  -H 'Content-type: application/json' \
  http://localhost:8003/v1/documents?uri=/example/hamlet.json
```

Then search for occurrences of SPEAKER with the value BERNARDO. The search matches the document just inserted and returns a snippet for the two speeches by Bernardo:

```
$ curl --anyauth --user user:password -X GET \
  'http://localhost:8003/v1/keyvalue?key=SPEAKER&value=BERNARDO'
...
{
```

```
  "snippet-format":"snippet",
  "total":1,
  "start":1,
  "page-length":10,
  "results":[{
    "index":1,
    "uri":"/example/hamlet.json",
    "path":"fn:doc(\"/example/hamlet.json\")",
    "score":154880,
    "confidence":0.790972,
    "fitness":0.790972,
    "matches":[
      {"path":"fn:doc(\"/example/hamlet.json\")/*:json/*:ACT/*:SCENE/*
:SPEECHES/*:json[1]/*:SPEECH/*:SPEAKER",
       "match-text":[{"highlight":"BERNARDO"}]},
      {"path":"fn:doc(\"/example/hamlet.json\")/*:json/*:ACT/*:SCENE/*
:SPEECHES/*:json[3]/*:SPEECH/*:SPEAKER",
       "match-text":[{"highlight":"BERNARDO"}]}
    ]
  }],
  "metrics":{
    "query-resolution-time":"PT0.001398S",
    "facet-resolution-time":"PT0.000052S",
    "snippet-resolution-time":"PT0.001411S",
    "total-time":"PT0.003032S"
  }
}
```

4.2.5 Searching By Element or Attribute Name and Value

The `/keyvalue` service provides a simple interface to match element and element attribute values in XML content. For more sophisticated element and element attribute value searches, use structured queries; see “Searching With Structured Queries” on page 133.

> **Note:** Efficient element and element attribute searches require element and attribute range indexes. For details, see Range Indexes and Lexicons in the *Administrator’s Guide*.

To find all XML elements with a specific element name and value, send a GET request to the `/keyvalue` service with a URL of the form:

```
http://host:port/version/keyvalue?element=elem_name&value=elem_value
```

> **Note:** If the element name is defined within a namespace, you must pre-define a namespace binding and use the defined prefix in *elem_name*. For details, see “Using Namespace Bindings” on page 177.

The following example matches documents containing a `<SPEAKER>` element with the value “BERNARDO”:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
   'http://localhost:8003/v1/keyvalue?element=SPEAKER&value=BERNARDO'
```

To find all XML elements with a specific element name that have an attribute with a specific value, send a GET request to the /keyvalue service with a URL of the form:

```
http://host:port/version/keyvalue?element=elem_name&attribute=attr_name&
value=elem_value
```

This example matches documents containing `<nominee>` elements with a `year` attribute value of 1946:

```
$ curl --anyauth --user user:password -X GET \
    'http://localhost:8003/v1/keyvalue?element=nominee&attribute=year&
value=1946'
```

MarkLogic Server can return search results in XML or JSON. Use the Accept header or `format` request parameter to select the output content type. For details, see “Controlling Input and Output Content Type” on page 25.

For a complete list of parameters available with the `/keyvalue` service, see “GET /v1/keyvalue” in the *REST Resources API*.

4.2.6 Debugging /search Queries With Logging

When you enable the `debug` REST API instance property and make queries using `/search`, query details are sent to the MarkLogic Server error log. Enable this logging by setting the debug property to true, as described in “Configuring Instance Properties” on page 41.

When you enable debug logging, requests to `/search` log the following information helpful in debugging queries:

- EFFECTIVE OPTIONS. These are the query options in effect for your query. If the request does not include an `options` request parameter or a combined query, these are the default options.
- CTS-QUERY. Your query, expressed as a `cts:query` object. A `cts:query` object is the low level XML representation of any query expression. For details, see Understanding cts:query in the *Search Developer’s Guide*.
- SEARCH-QUERY. Your query, expressed as a `search:query` object. This information is only included if your request includes a structured query. A `search:query` is the XML representation of a structured query. For details, see Searching Using Structured Queries in the *Search Developer’s Guide*.

Examining the logging output can help you understand how MarkLogic Server sees your query. In the following example, notice that a string query for "Welles" is really a `cts:word-query` for the text string "Welles". With the addition of a structured query in the request body to limit the query to ocurrences of "Welles" to documents that also include "John" when it occurs near "Huston", the logging output includes a `cts:query` and a `search:query` that reflect the result of combining the string query and structured query.

1. Enable debug logging for your REST API instance.

```
$ cat debug-on.xml
<properties xmlns="http://marklogic.com/rest-api">
  <debug>true</debug>
</properties>

$ curl --anyauth  --user user:password -X PUT \
    -d @./debug-on.xml -H "Content-type: application/xml" \
    http://localhost:8003/v1/config/properties
```

2. Issue a query to `/search`.

```
$ curl --anyauth --user user:password -X GET \
    -H "Accept: application/xml" \
    http://localhost:8003/v1/search?q="Welles"
...search matches returned
```

3. View the debug output in the MarkLogic Server error log, located in *MARKLOGIC_DIR*`/Logs/ErrorLog.txt`. There is no `search:query` in the log because this is a simple string query. (The timestamp and Info: header on each log line have been elided).

```
$ tail -15 /var/opt/MarkLogic/Logs/ErrorLog.txt
... Request environment:
... GET /v1/search?q=Douglas
... Rewritten to:
    /MarkLogic/rest-api/endpoints/search-list-query.xqy?q=Douglas
... ACCEPT */*
... PARAMS:
...   q: (Welles)
...
... Endpoint Details:
... EFFECTIVE OPTIONS:
... <options xmlns="http://marklogic.com/appservices/search">
...   <search-option>unfiltered</search-option>
...   <quality-weight>0</quality-weight>
... </options>
... CTS-QUERY:
... <cts:word-query qtextref="cts:text"
            xmlns:cts="http://marklogic.com/cts">
      <cts:text>Welles</cts:text>
    </cts:word-query>
```

4. Add a structured query to the search to constrain matches to documents that also contain "John" occurring near "Huston".

```
$ cat ./structq.json
{"query":
  {"near-query":
    {"queries":[
      {"term-query":{"text":"John"}},
      {"term-query":{"text":"Huston"}}
    ]}
  }
}

$ curl --anyauth --user user:password -X POST -d@./structq.json \
    -H "Content-type: application/json" \
    http://localhost:8003/v1/search?q=Welles
...search results returned
```

5. View the logging output again and notice the inclusion of a search:query section in the log. Also, notice that the `cts:query` and the `search:query` represent the combined string and structured queries.

```
... Request environment:
... POST /v1/search?q=Welles
... Rewritten to:
    /MarkLogic/rest-api/endpoints/search-list-query.xqy?q=Welles
... ACCEPT */*
... PARAMS:
...   q: (Welles)
...
... Endpoint Details:
... EFFECTIVE OPTIONS:
... <options xmlns="http://marklogic.com/appservices/search">
...   <search-option>unfiltered</search-option>
...   <quality-weight>0</quality-weight>
... </options>
... CTS-QUERY:
... cts:and-query(
      (
        cts:word-query("Welles", ("lang=en"), 1),
        cts:near-query((
          cts:word-query("John", ("lang=en"), 1),
          cts:word-query("Huston", ("lang=en"), 1)),
        10, (), 1)
       ), ()
     )
... SEARCH-QUERY:
... <search:query
          xmlns:search="http://marklogic.com/appservices/search">
...   <search:qtext>Welles</search:qtext>
...   <search:near-query>
...     <search:term-query>
...       <search:text>John</search:text>
```

```
...      </search:term-query>
...      <search:term-query>
...        <search:text>Huston</search:text>
...      </search:term-query>
...    </search:near-query>
... </search:query>
```

4.3 Querying Lexicons and Range Indexes

The `/values` service supports the following operations:

- Query the values in a single lexicon or range index, or
- Find co-occurrences of values in multiple range indexes
- Analyze range index values or value co-occurrences using builtin or user-defined aggregate functions. For details, see "Analyzing Lexicons and Range Indexes With Aggregate Functions" on page 156.

For related search concepts, see Browsing With Lexicons in the *Search Developer's Guide*.

This section covers the following topics:

- Querying the Values in a Lexicon or Range Index
- Finding Value Co-Occurrences in Lexicons
- Using a Query to Constrain Results
- Identifying Lexicon and Range Index Values in Query Options
- Creating Indexes on JSON Keys
- Limiting the Number of Results

4.3.1 Querying the Values in a Lexicon or Range Index

Use the `/values/{name}` service to query the values in a lexicon or range index. Such queries must be supported by query options that include a `<values/>` element identifying the target lexicon or index; for details, see "Defining Queryable Lexicon or Range Index Values" on page 147.

To query the values in a lexicon or range index, use the `/values/{name}` service as follows:

1. Install query options or define dynamic query options that include a `values` option naming the target lexicon or index. For details, see "Adding Query Options to a Request" on page 168.

2. Send a GET or POST request to the `/values/{name}` service, where *`name`* is the name of a `values` definition in the query options from Step 1 or in the default query options. If you use persistent query options, include the `options` parameter and replace *`options_name`* with the name under which the query options are installed.

```
http://host:port/version/values/name?options=options_name
```

When constructing your request:

1. Substitute a named range specification for *name* in the URL. This must be a `values` range specification in the query options named by the `options` parameter, in the `options` portion of a combined query in the POST body, or in the default query options.

2. To use custom query options, specify them using the `options` parameter and/or the `options` portion of a combined query in the POST body. For details, see "Identifying Lexicon and Range Index Values in Query Options" on page 147 and "Adding Query Options to a Request" on page 168.

3. To use the default query options, omit the `options` parameter and `options` portion of a combined query. The default query options should include a range specification matching *name*.

4. To constrain the analysis to values in certain fragments, specify a query using the `q` and/or `structuredQuery` parameters, or a structured or combined query in the POST body. For details, see "Using a Query to Constrain Results" on page 146.

5. Specify the input (POST only) and output content type (XML or JSON) using the `format` parameter or the HTTP Content-type and Accept headers. For details, see "Controlling Input and Output Content Type" on page 25.

Additional request parameters are available. For details, see the *MarkLogic REST API Reference*.

The following example assumes the query options shown in the table below are installed under the name `index-options`:

Format	Query Options
XML	<options xmlns="http://marklogic.com/appservices/search"> <values name="speaker"> <range type="xs:string"> <element ns="" name="SPEAKER"/> </range> </values> </options>

JSON	{
	  "options": {
	  "values": [
	  {
	  "name": "speaker",
	  "range": {
	  "type": "xs:string",
	  "element": { "ns": "", "name": "SPEAKER" }
	  }
	  }
	  ]
	  }
	}

Then the example command below queries `/values/speaker` to retrieve the values of all SPEAKER elements:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    http://localhost:8003/v1/values/speaker?options=index-options
...
<values-response name="speaker" type="xs:string" \
    xmlns="http://marklogic.com/appservices/search" \
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <distinct-value frequency="1">[GOWER]</distinct-value>
  <distinct-value frequency="1">[PROSPERO]</distinct-value>
  ...
</values-response>
```

If you use the `format` parameter to request JSON output from the same request, the results are similar to the following:

```
{
  "values-response": {
    "name": "speaker",
    "type": "xs:string",
    "distinct-value": [
      {
        "frequency": 1,
```

```
        "_value": "[GOWER]"
      },
      {
        "frequency": 1,
        "_value": "[PROSPERO]"
      },
      ...
    ],
    "metrics": {
      "values-resolution-time": "PT0.016665S",
      "aggregate-resolution-time": "PT0.00001S",
      "total-time": "PT0.018102S"
    }
  }
}
```

If you add a string query, structured query, or both to the request, you can limit the results to index values in matching fragments. For example, the following requests returns only the SPEAKER values in fragments containing “HAMLET”:

```
$ curl --anyauth --user user:password -X GET \
    'http://localhost:8003/v1/values/speaker?options=index-options&q="
HAMLET"'
```

4.3.2 Finding Value Co-Occurrences in Lexicons

A co-occurence is a set of index or lexicon values occurring in the same document fragment. The MarkLogic REST API enables you to query for *n*-way co-occurrences. That is, tuples of values from multiple lexicons or indexes, occurring in the same fragment.

Use this procedure to query for co-occurrences of values in lexicons or range indexes with the `/values/{name}` service:

1. Specify query options that include a `tuples` range specification for the target lexicons or indexes. For details, see “Defining Queryable Lexicon or Range Index Co-Occurrences” on page 149 and “Adding Query Options to a Request” on page 168.

2. Send a GET or POST request of the following form to the `/values/{name}` service, where *`name`* is the name of a `tuples` definition in the query options from Step 1. If you use persistent query options, include the `options` parameter and replace *`options_name`* with the name under which the query options are installed.

   ```
   http://host:port/version/values/name?options=options_name
   ```

When constructing your request:

1. Substitute a named range specification for *`name`* in the URL. This must be a `tuples` range specification in the query options named by the `options` parameter, in the `options` portion

of a combined query in the POST body, or in the default query options if `options` is omitted.

2. To use custom query options, specify them using the `options` parameter and/or the `options` portion of a combined query in the POST body. For details, see "Identifying Lexicon and Range Index Values in Query Options" on page 147 and "Adding Query Options to a Request" on page 168.

3. To use the default query options, omit the `options` parameter and `options` portion of a combined query. The default query options should include a range specification matching *`name`*.

4. To constrain the analysis to values in certain fragments, specify a query using the `q` and/or `structuredQuery` parameters, or a structured or combined query in the POST body. For details, see "Using a Query to Constrain Results" on page 146.

5. Specify the input (POST only) and output content type (XML or JSON) using the `format` parameter or the HTTP Content-type and Accept headers. For details, see "Controlling Input and Output Content Type" on page 25.

Additional request parameters are available. For details, see the *MarkLogic REST API Reference*.

For more information about co-occurrences, see Value Co-Occurrences Lexicons in the *Search Developer's Guide*.

The following example assumes the query options shown in the table below are installed under the name `index-options`. Note that the options include a `<tuples/>` definition named "speaker-scene".

Format	Query Options
XML	<options xmlns="http://marklogic.com/appservices/search"> <tuples name="speaker-scene"> <range type="xs:string"> <element ns="" name="SPEAKER"/> </range> <range type="xs:string"> <path-index> /PLAY/ACT/SCENE/TITLE </path-index> </range> </tuples> </options>
JSON	{ "options": { "tuples": [{ "name": "speaker-scene", "range": [{ "type": "xs:string", "element": { "ns": "", "name": "SPEAKER" } }, { "type": "xs:string", "path-index": { "text": "\/PLAY\/ACT\/SCENE\/TITLE" } }] }] } }

Given these query options, this example queries `/values/speaker-scene` to retrieve co-occurrences of `SPEAKER` and scene titles:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
   http://localhost:8003/v1/values/speaker-scene?options=index-options
...
<values-response name="speaker-scene"
    xmlns="http://marklogic.com/appservices/search"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
  <tuple frequency="1">
    <distinct-value xsi:type="xs:string" ...>A Lord</distinct-value>
    <distinct-value ...>SCENE II.  The forest.</distinct-value>
  </tuple>
  ...
</values-response>
```

If you add a string query, structured query, or both to the request, you can limit the results to co-occurrences in matching fragments. For example, the following requests returns only the `SPEAKER` values in fragments containing “HAMLET”:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    'http://localhost:8003/v1/values/speaker-scene?options=index-optio
ns&q="HAMLET"'
```

4.3.3 Using a Query to Constrain Results

You can use a string query, structured query, or combined query with `/values/{name}` to limit results to fragments that match the query. The values must occur in fragments matching the query. The fragments are selected in the same manner as an “unfiltered” `cts:search`; for details, see Understanding Unfiltered Searches in the *Query Performance and Tuning Guide*.

If you use a string query, the query is treated as a word query; for details, see `cts:word-query`. Supply a string query using one of the following:

- The `q` request parameter of a POST or GET request
- The `qtext` element or or sub-object of a combined query supplied in the body of a POST request.

Specify a structured query in either XMl or JSON using one of the following:

- The `structuredQuery` request parameter on a GET request.
- The `query` element or sub-object of a combined query supplied in the body of a POST request.
- In the body of a POST request that does not use a combined query.

The following example limits the results to just those fragments containing the word “moon”:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    'http://localhost:8003/v1/values/speaker?options=index-options&q=m
oon'
```

For the full example without a query constraint, see “Querying the Values in a Lexicon or Range Index” on page 140.

For details on query syntax, see the following sources:

- Searching Using String Queries in the *Search Developer's Guide*
- Searching Using Structured Queries in the *Search Developer's Guide*
- "Specifying Dynamic Query Options with Combined Query" on page 161

4.3.4 Identifying Lexicon and Range Index Values in Query Options

Before you can use the `/values/{name}` service, you must install query options that identify lexicons and range indexes available to queries. Use the `values` option to make the values in a single lexicon or index available. Use the `tuples` option to make co-occurrences of values in multiple lexicons or indexes available.

This section covers the following topics:

- Defining Queryable Lexicon or Range Index Values
- Defining Queryable Lexicon or Range Index Co-Occurrences

4.3.4.1 Defining Queryable Lexicon or Range Index Values

Use this procedure to make the values in a lexicon or range index available through the `/values/{name}` service:

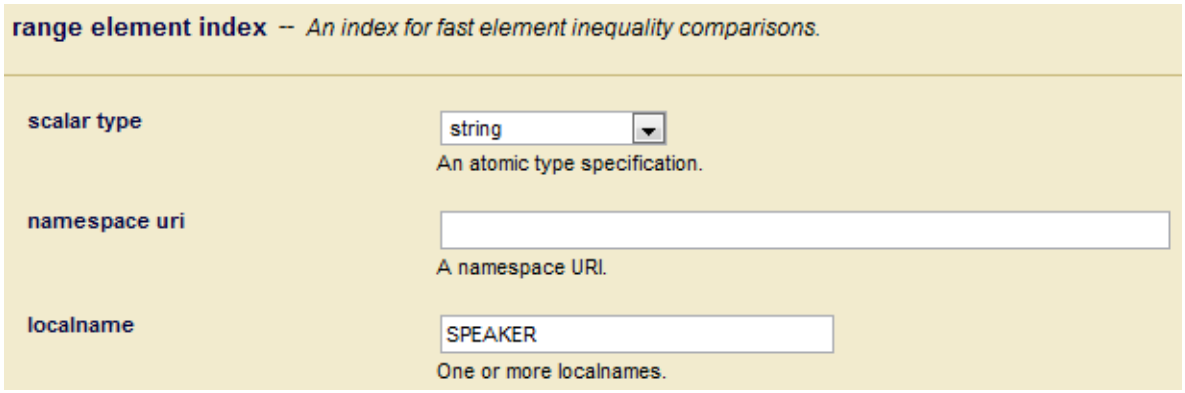
1. Create a lexicon or range index on the database, as described in Range Indexes and Lexicons in the *Administrator's Guide* and

2. Associate a name with the index or lexicon by defining a `<values/>` element (or JSON sub-object) in query options.

3. Supply the query options from Step 2 in your request, as described in "Adding Query Options to a Request" on page 168.

For more information on lexicons and range indexes, see Browsing With Lexicons in the *Search Developer's Guide* and "Creating Indexes on JSON Keys" on page 150.

For example, if the database configuration includes an element range index on SPEAKER, such as the one shown below:



Then the following query options enable the `SPEAKER` element values to be referenced using as the resource `/values/speaker`, once you install the options:

Format	Query Options

XML	`<options xmlns="http://marklogic.com/appservices/search">`
	`  <values name="speaker">`
	`  <range type="xs:string">`
	`  <element ns="" name="SPEAKER"/>`
	`  </range>`
	`  </values>`
	`</options>`

JSON	`{`
	`  "options": {`
	`  "values": [`
	`  {`
	`  "name": "speaker",`
	`  "range": {`
	`  "type": "xs:string",`
	`  "element": { "ns": "", "name": "SPEAKER" }`
	`  }`
	`  }`
	`  ]`
	`  }`
	`}`

The following example command installs the XML query options with the name “index-options”:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT \
    -d@"./index-options.xml" -H "Content-type: application/xml" \
    http://localhost:8003/v1/config/query/index-options
```

Now, you can query the `SPEAKER` index using the `/values/speaker` resource with the query options named “index-options”.

4.3.4.2 Defining Queryable Lexicon or Range Index Co-Occurrences

A `<tuples/>` element in query options specifies the indexes to use in constructing *n*-way value co-occurrences. For more information about co-occurrences, see Value Co-Occurrences Lexicons in the *Search Developer's Guide*.

Use this procedure to make co-occurrences of values in multiple lexicons or range indexes available through the `/values/{name}` service:

1. Create the lexicons or range indexes on the database, as described in Range Indexes and Lexicons in the *Administrator's Guide*.

2. Associate a name with the lexicon/index tuple by defining a `<tuples>` element in query options (or a “tuples” object in JSON).

3. Supply the query options from Step 2 in your request, as described in “Adding Query Options to a Request” on page 168.

For example, suppose the database configuration contains a string-valued element range index on `<SPEAKER>` and a path range index on the XPath expression `/PLAY/ACT/SCENE/TITLE`. The following query options enable querying co-occurrrences of these two indexes under the name “speaker-scene”:

Format	Query Options
XML	<options xmlns="http://marklogic.com/appservices/search"> <tuples name="speaker-scene"> <range type="xs:string"> <element ns="" name="SPEAKER"/> </range> <range type="xs:string"> <path-index>/PLAY/ACT/SCENE/TITLE</path-index> </range> </tuples> </options>

JSON	{
	  "options": {
	  "tuples": {
	  "name": "speaker-scene",
	  "range": {
	  "type": "xs:string",
	  "path-index": {
	  "text": "/PLAY/ACT/SCENE/TITLE",
	  }
	  }
	  }
	  }
	}

The following example command installs the XML query options with the name “index-options”:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT \
    -d@"./index-options.xml" -H "Content-type: application/xml" \
    http://localhost:8003/v1/config/query/index-options
```

Now, you can query the SPEAKER index using the `/values/speaker-scene` resource with the query options named “index-options”.

4.3.5 Creating Indexes on JSON Keys

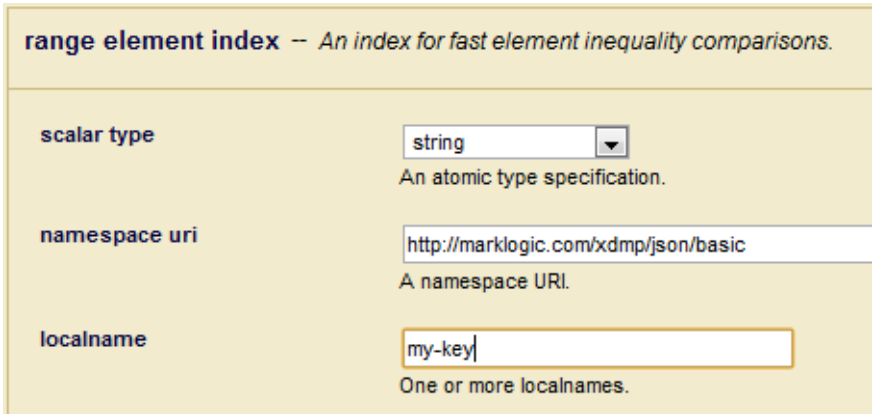
To efficiently search using JSON keys, you should define indexes on the keys. For example, using `/keyvalue` to query by JSON key values performs best when you define a value range index on the key you want to query against. To create such an index, you treat the JSON key as an XML element and define a corresponding element index, as described here.

JSON documents loaded using the MarkLogic REST API are stored internally as XML. Each JSON key has a corresponding XML element in the internal representation. The element name usually matches the JSON key name, but can be different if the key contains whitespace or other characters that cannot appear in an XML element name. Key names are encoded using the XQuery builtin `xdmp:encode-for-NCName`.

> **Note:** Element names in the XML representation are in the namespace "http://marklogic.com/xdmp/json/basic". You must use this namespace when defining indexes on keys in JSON documents.

For more information on JSON support in MarkLogic Server, see Working With JSON in the *Application Developer's Guide*.

To create an index on a JSON key, you must determine the XML element name that corresponds to the key in the XML representation of your JSON content, and then create an element range index on that element. For example, to efficiently search the values of the JSON key "my-key", use the Admin Interface to create a range index with the following settings:



4.3.6 Limiting the Number of Results

You can use the `limit`, `start`, and `pageLength` request parameters to limit the number of values or co-occurrences returned by GET or POST /v1/values.

The `limit` parameter specifies the maxiumum number of value to retrieve from a lexicon. Use `start` and `pageLength` to return results one page at a time, similar to the way you can page through results from the `/search` service. If `limit` is present, then `start` and `pageLength` are applied to the subset of values selected by `limit`, so the values on a page never extend beyond the values selected by `limit`.

For example, in the following request, at most 2 values or tuples are returned because `start` + `pageLength` would extend beyond the 5 values selected by `limit`.

```
GET /v1/values?limit=5&start=4&pageLength=3
```

If you specify a `start` value, you must also specify a `pageLength`. For a detailed discussion of how these parameters affect the results returned by values requests, see Returning Lexicon Values With search:values in the *Search Developer's Guide*.

4.4 Using Query By Example to Prototype a Query

This section describes how to search XML and JSON documents using a Query By Example (QBE). You cannot use QBE to search other document types or to search metadata.

This section covers the following topics:

- What is QBE
- Searching Documents With QBE
- Validating a QBE
- Generating a Combined Query from a QBE

4.4.1 What is QBE

A Query By Example (QBE) enables rapid prototyping of queries for "documents that look like this" using search criteria that resemble the structure of documents in your database.

For example, if your documents include an `author` element or key, you can use the following QBE to find documents with an `author` value of "Mark Twain".

Format	Example

XML	`<q:qbe xmlns:q="http://marklogic.com/appservices/querybyexample">`
	`  <q:query>`
	`  <author>Mark Twain</author>`
	`  </q:query>`
	`</q:qbe>`

JSON	`{`
	`  "$query": { "author": "Mark Twain" }`
	`}`

You can only use QBE to search XML and JSON documents. Metadata search is not supported. You can search by element, element attribute, and JSON key; fields are not supported. For structural details, see Searching Using Query By Example in *Search Developer's Guide*.

When you're satisfied with your prototype or ready to use more powerful Search API features, you can use the API to convert a QBE into a combined query for use with the `/search` service.

The MarkLogic REST API includes the following support for QBE through the `/qbe` service:

- Search XML and JSON documents using a QBE.
- Validate the correctness of a QBE.
- Convert a QBE to a combined query for improved performance and full expressiveness.

4.4.2 Searching Documents With QBE

To search using QBE, send a GET or POST request to the `/qbe` service. To pass the QBE as a request parameter, send a GET request with a URL of the form:

```
http://host:port/version/qbe?query=your-qbe
```

Where *your-`qbe`* is an XML or JSON representation of a QBE.

To pass a QBE in the request body, send a POST request of the following form and place a QBE in the POST body. Set the Content-type header appropriately.

```
http://host:port/version/qbe
```

You can request search results in XML or JSON. To control the output content type, use the `format` request parameter or `Accept` headers. If you use the `format` parameter on a POST request, the input and output format must be the same. For details, see “Controlling Input and Output Content Type” on page 25.

You can validate the correctness of your input QBE as part of your search or as a standalone operation. For details, see “Validating a QBE” on page 154.

For a complete list of parameters available with the `/qbe` service, see `GET:/v1/qbe` or `POST:/v1/qbe` in the *REST Resources API*.

The following example matches XML documents that have an author element value of “Mark Twain”. Results are returned as XML.

```
$ cat qbe.xml
<q:qbe xmlns:q="http://marklogic.com/appservices/querybyexample">
  <q:query>
    <author>Mark Twain</author>
  </q:query>
</q:qbe>

# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X POST -d@"./qbe.xml" \
  -H 'Content-type: application/xml' \
  'http://localhost:8003/v1/qbe'
```

The following example shows an equivalent search using JSON.

```
$ cat qbe.sjon
{"$query": {
```

```
    "author": "Mark Twain"
} }

# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X POST -d@"./qbe.json" \
  -H 'Content-type: application/json' \
  -H 'Accept: application/json' \
  'http://localhost:8003/v1/qbe
```

The `/qbe` service supports most of the same features as the `/search` service, such as using pre-installed persistent query options, result pagination, and search result transformations. For details, see `GET:/v1/qbe` or `POST:/v1/qbe` in the *REST Resources API*.

You can also use `/qbe` to retrieve matching documents and metadata. The examples in this section only return a search response. To retrieve whole documents and/or their metadata, see "Reading Multiple Documents Matching a Query" on page 240.

4.4.3 Validating a QBE

When you perform a search, MarkLogic Server does not verify the correctnesss of your QBE. If your QBE is syntactically or semantically incorrect, you might get errors or surprising results. To avoid such issues, you can validate your QBE prior to or as part of a search.

To validate your query as a standalone operation, add the request parameter `view=validate` to a GET or POST request to the `/qbe` service. Rather than performing a search, MarkLogic Server checks your QBE for correctness and returns an indication of validity returned.

For example, the following command validates a JSON QBE:

```
$ curl --anyauth --user user:password -i -X POST \
    -H "Content-type: application/json" -d @./qbe.json \
    'http://localhost:8003/v1/qbe?view=validate'
```

If your query is valid, MarkLogic Server responds with status 200 (OK) and the response body contains a `valid-query` element, similar to the following:

```
<q:valid-query
  xmlns:q="http://marklogic.com/appservices/querybyexample"/>
```

If your query is invalid, MarkLogic Server can respond with either a status 200 (OK) or status 400 (Bad Request), depending on the nature of the error. When the status code is 400, the response body contains an error. When the status code is 200, the response body contains an `invalid-query` element that encapsulates the reason validation failed. For example:

```
<q:invalid-query
    xmlns:q="http://marklogic.com/appservices/querybyexample">
  <q:report id="QBE-QUERY">Query can only contain a filtered flag,
score configuration,  composers, word queries, and criteria</q:report>
</q:invalid-query>
```

To validate a query as part of your search, use the request parameters `view=validate` in conjunction with `view=results`. If your query is valid, the search proceeds as usual. If you query is not valid, an error report is returned.

```
$ curl --anyauth --user user:password -i -X POST \
    -H "Content-type: application/json" -d @./qbe.json \
    'http://localhost:8003/v1/qbe?view=validate&view=results'
```

4.4.4 Generating a Combined Query from a QBE

Generating a combined query from a QBE has the following potential benefits:

- Improve search performance.
- Access a wider array of search features.
- Debug your QBE by examining the lower level Search API constructs it generates.

To generate a combined query from a QBE, add the `view=structured` request parameter to a GET or POST request to the `/qbe` service. Rather than performing a search, the request returns a combined query in the response. You can use the resulting query with the `/search` service.

You cannot combine `view=structured` with other `view` settings, such as `validate` or `results`.

The following command generates a combined query from a QBE:

```
$ cat qbe.xml
<q:qbe xmlns:q="http://marklogic.com/appservices/querybyexample">
  <q:query>
    <author>Mark Twain</author>
  </q:query>
</q:qbe>

# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X POST -d@"./qbe.xml" \
  -H 'Content-type: application/xml' \
  'http://localhost:8003/v1/qbe?view=structured'

HTTP/1.1 200 OK
...
<search:search xmlns:search="http://marklogic.com/appservices/search">
  <search:query>
    <search:value-query>
      <search:element ns="" name="author"/>
      <search:text>Mark Twain</search:text>
      <search:term-option>exact</search:term-option>
    </search:value-query>
  </search:query>
  <search:options>
    <search:search-option>unfiltered</search:search-option>
    <search:quality-weight>0</search:quality-weight>
    <search:result-decorator apply="href-decorator"
```

```
        ns="http://marklogic.com/rest-api/lib/href-decorator"
        at="/MarkLogic/rest-api/lib/rest-result-decorator.xqy"/>
    </search:options>
</search:search>
```

For more details, see “Searching Using Structured Queries” on page 71 in *Search Developer’s Guide* and “Specifying Dynamic Query Options with Combined Query” on page 161.

4.5 Analyzing Lexicons and Range Indexes With Aggregate Functions

This section covers the following topics:

- Aggregate Function Overview
- Using Query Options to Apply Aggregate Functions
- Using Request Parameters to Apply Aggregate Functions
- Example: Applying a Builtin Aggregate Function
- Example: Applying an Aggregate UDF

4.5.1 Aggregate Function Overview

An aggregate function performs an operation over values or value co-occurrences in lexicons and range indexes. For example, you can use an aggregate function to compute the sum of values in a range index.

There are two kinds of aggregate functions, builtin and user-defined. MarkLogic Server provides builtin aggregate functions for several common analytical functions; see the list in Using Builtin Aggregate Functions in the *Search Developer’s Guide*.

In addition, you can also implement aggregate user-defined functions (UDFs) in C++ and deploy them as native plugins. Aggregate UDFs must be installed before you can use them. For details, see Implementing an Aggregate User-Defined Function in the *Application Developer’s Guide*.

You can use the MarkLogic REST API to apply aggregate functions using `/values/{name}` in two ways:

- Include one or more `<aggregate/>` elements (XML) or sub-objects (JSON) in a `<values/>` or `<tuples/>` range specification in the query options.
- Include one or more `aggregate` request parameters.

If aggregate functions are specified through both query options and request parameters, the request parameter(s) overrides the aggregates specified in the query options.

> **Note:** You can only specify multiple aggregate UDFs from more than one plugin using query options.

Note: You cannot use the MarkLogic REST API to apply aggregate UDFs that require additional parameters.

4.5.2 Using Query Options to Apply Aggregate Functions

To specify an aggregate function in query options, include an `<aggregate/>` element in a `<values/>` or `<tuples/>` range specification. If you include multiple `<aggregate/>` specifications, MarkLogic Server applies all the functions.

For a builtin aggregate, specify the function name in the "apply" attribute of an `<aggregate/>` element. For example, the query options below specify the builtin aggregate "count", which is equivalent to the XQuery builtin `cts:count-aggregate`.

Format	Query Options

XML	`<options xmlns="http://marklogic.com/appservices/search">`
	`  <values name="speaker">`
	`  <range type="xs:string">`
	`  <element ns="" name="SPEAKER"/>`
	`  </range>`
	`  **<aggregate apply="count"/>**`
	`  </values>`
	`</options>`

JSON	`{`
	`  "options": {`
	`  "values": [`
	`  {`
	`  "name": "speaker",`
	`  "range": {`
	`  "type": "xs:string",`
	`  "element": { "ns": "", "name": "SPEAKER" }`
	`  },`
	`  **"aggregate": { "apply":"count" }**`
	`  }`
	`  ]`
	`  }`
	`}`

An aggregate UDF is identified by the function name and a relative path to the plugin that implements the aggregate, as described in Using Aggregate User-Defined Functions in the *Search Developer's Guide*. Specify the function name with the "apply" attribute and the plugin path with the "udf" attribute in an `<aggregate/>` element or object. For example, the following query options specify a native UDF called "count" provided by a plugin installed under "native/sampleplugin":

Format	Query Options
XML	<options xmlns="http://marklogic.com/appservices/search"> <values name="speaker"> <range type="xs:string"> <element ns="" name="SPEAKER"/> </range> **<aggregate apply="count" udf="native/sampleplugin" />** </values> </options>

JSON	{
	  "options": {
	  "values": [
	  {
	  "name": "speaker",
	  "range": {
	  "type": "xs:string",
	  "element": { "ns": "", "name": "SPEAKER" }
	  },
	  "aggregate": {
	  "apply":"count",
	  "udf":"native/sampleplugin"
	  }
	  }
	  ]
	  }
	}

To use query options to apply an aggregate function:

1. Define query options that include one or more aggregate definitions, as shown above.

2. Supply the query options from Step 1 in your request, as described in"Adding Query Options to a Request" on page 168.

3. Apply the aggregate by sending a GET or POST request to `/values/{name}` and including the options from Step 2. For example: `GET /v1/values/speaker?options=index-options`.

For details on using query options, see "Configuring Query Options" on page 167.

4.5.3 Using Request Parameters to Apply Aggregate Functions

To analyze lexicon or index values or co-occurrences with builtin aggregate function, make a GET or POST request to the `/values/{name}` service with a URL of the form:

```
http://host:port/version/values/name?aggregate=aggr_name&options=options_name
```

To analyze lexicon or index values or co-occurrences with a previously installed aggregate UDF, make a GET request to the `/values/{name}` service with a URL of the form:

```
http://host:port/version/values/name?aggregate=aggr_name&aggregatePath
=aggr_path&options=options_name
```

When constructing the request:

1. Substitute a named range specification for *name* in the URL. This must be a range specification (`<values/>` or `<tuples/>`) in the query options supplied in the request, or in the default query options if options are omitted.

2. To use custom query options, specify them using the `options` parameter and/or the `options` portion of a combined query in the POST body. For details, see "Adding Query Options to a Request" on page 168.

3. To use the default query options, omit the `options` parameter and `options` portion of a combined query. The default query options should include a range specification matching *name*.

4. Specify the aggregate function name using the `aggregate` parameter.

   The name must be one of the builtin aggregate functions listed in Using Builtin Aggregate Functions the *Search Developer's Guide*, or a function implemented by the plugin identified by `aggregatePath`.

5. If you're applying an aggregate UDF, specify the relative path to the plugin implementing the aggregate function using the `aggregatePath` parameter.

6. To constrain the analysis to values in certain fragments, specify a query using the `q` and/or `structuredQuery` parameters, or a structured or combined query in the POST body. For details, see "Using a Query to Constrain Results" on page 146.

7. Specify the result content type (XML or JSON) using the `format` parameter or the HTTP Accept headers. The default format is XML. For details, see "Controlling Input and Output Content Type" on page 25.

8. If you only want the aggregate value in the results, set the `view` parameter to `aggregate`. By default, MarkLogic Server returns both the lexicon or index values and the aggregate result.

Additional request parameters are available. For details see the *MarkLogic REST API Reference*.

When applying an aggregate UDF, the output is dependent on the UDF. Aggregate UDFs return a sequence of items, which can be atomic values or key-value maps. For details, see User-Defined Functions in the *Application Developer's Guide* and the XQuery builtin function `cts:aggregate`.

4.5.4 Example: Applying a Builtin Aggregate Function

The following example counts the number of values in the index identified as “speaker” in “index-options” (`options=index-options`). The counted values include only those in fragments containing “HAMLET” (`q=HAMLET`). The output should contain only the aggregate result (`view=aggregate`).

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    'http://localhost:8003/v1/values/speaker?options=index-options&agg
regate=count&view=aggregate&q=HAMLET'
...
<values-response name="speaker" type="xs:string"
    xmlns="http://marklogic.com/appservices/search"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <aggregate-result name="count">92</aggregate-result>
  <metrics>
    <aggregate-resolution-time>PT0.001108S</aggregate-resolution-time>
    <total-time>PT0.003719S</total-time>
  </metrics>
</values-response>
```

4.5.5 Example: Applying an Aggregate UDF

This example demonstrates using an aggregate user-defined function to count the number of values in an element range index using `/values/{name}`. The aggregate UDF is specified via request parameters, as described in “Using Request Parameters to Apply Aggregate Functions” on page 158.

This example assumes the following pre-requisites are already met:

- A native plugin is installed with the path “native/sampleplugin”.
- The plugin implements a “count” function that counts the values in a range index. That is, a function equivalent to the “count” builtin aggregate function.
- Query options are installed with the name “index-options”.
- The query options include a range specification named “speaker” for the SPEAKER element range index.

The following command uses the “count” aggregate UDF to count the number of values in the SPEAKER index in fragments containing “HAMLET”. The output only contains the aggregate result (`view=aggregate`).

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    'http://localhost:8003/v1/values/speaker?options=index-options&agg
regate=count&aggregatePath=native/sampleplugin&view=aggregate&q=HAMLET
'
```

The use of `view=aggregate` limits the output to only the aggregate results, as shown below. XML is the default output format. Use the `format` parameter or the HTTP Accept headers to request JSON output.

Format	Example Output

XML	`<values-response name="speaker" type="xs:string"`
	`  xmlns="http://marklogic.com/appservices/search"`
	`  xmlns:xs="http://www.w3.org/2001/XMLSchema">`
	`  **<aggregate-result name="count">92</aggregate-result>**`
	`  <metrics>`
	
	`<aggregate-resolution-time>PT0.001514S</aggregate-resolution-time>`
	`  <total-time>PT0.004049S</total-time>`
	`  </metrics>`
	`</values-response>`

JSON	`{`
	`  "values-response": {`
	`  "name": "speaker",`
	`  "type": "xs:string",`
	`  **"aggregate-result": [**`
	`  **{**`
	`  **"name": "count",**`
	`  **"_value": "92"**`
	`  **}**`
	`  ],`
	`  "metrics": {`
	`  "aggregate-resolution-time": "PT0.001336S",`
	`  "total-time": "PT0.004104S"`
	`  }`
	`  }`
	`}`

4.6 Specifying Dynamic Query Options with Combined Query

A *combined query* is an XML or JSON wrapper around a string and/or structured query and query options. Use a combined query to specify query options on the fly, without first persisting them as named options using the `/config/query` service. Combined queries are useful for rapid prototying during development, and for applications that need to modify query options on a per query basis.

To use a combined query, send a POST request to `/v1/search` or `/v1/values/{name}` with the combined query in the request body. See the following topics for more details:

- Syntax
- Interaction with Queries in Request Parameters
- Interaction with Persistent Query Options

- Performance Considerations
- Combined Query Examples

4.6.1 Syntax

A combined query can contain a string query, a structured query, query options, or any combination of these. For example, you can create a combined query that contains only query options, only a structured query, or a structured query, string query and query options.

The following table shows the structure of a combined query. Within the combined query wrapper, the queries and options use the same syntax as when they occur standalone.

Format	Combined Query Template

XML	`<search xmlns="http://marklogic.com/appservices/search">`
	`  <query>`
	`  <!-- structured query, same syntax as standalone -->`
	`  </query>`
	`  <qtext>your string query</qtext>`
	`  <options>`
	`  <!-- same syntax as standalone query options -->`
	`  </options>`
	`</search>`

JSON	`{"search": {`
	`  "query": { structured query, same syntax as standalone },`
	`  "qtext": "your string query here"`
	`  "options": { same syntax as standalone query options }`
	`} }`

For details on sub-query and query options syntax, see the following sections of the *Search Developer's Guide*:

- Searching Using String Queries
- Searching Using Structured Queries
- Search Customization Using Query Options

4.6.2 Interaction with Queries in Request Parameters

When making a POST request to `/v1/search` or `/v1/values/{name}`, you can use a combined query in conjunction with the `q` request parameter. The string query in the request parameter value is AND'd with the sub-query(s) in the combined query.

The following table summarizes the interaction between the `q` request parameter and the sub-queries of a combined query. Note that the "queries" in the table are an abstraction rather than actual example queries.

Request Parameter	Combined Query	Final Query
q=*query-rp*	<search> <query>*query-cq*</query> </search>	*query-rp* AND *query-cq*
q=*query-rp*	<search> <qtext>*query-cq*</qtext> </search>	*query-rp* AND *query-cq*
q=*query-rp*	<search> <query>*query-cq1*</query> <qtext>*query-cq2*</qtext> </search>	*query-rp* AND *query-cq1* AND *query-cq2*
none	<search> <query>*query-cq1*</query> <qtext>*query-cq2*</qtext> </search>	*query-cq1* AND *query-cq2*

4.6.3 Interaction with Persistent Query Options

Dynamic query options supplied in a combined query are merged with persistent and default options that are in effect for the search. If the same non-constraint option is specified in both the combined query and persistent options, the setting in the combined query takes precedence.

Constraints are overridden by name. That is, if the dynamic and persistent options contain a `<constraint/>` element with the same `@name`, the definition in the dynamic query options is the one that applies to the query. Two constraints with different name are both merged into the final options.

```
<options xmlns="http://marklogic.com/appservices/search">
  <fragment-scope>properties</fragment-scope>
  <return-metrics>false</return-metrics>
  <constraint name="same">
    <collection prefix="http://server.com/persistent/"/>
  </constraint>
  <constraint name="not-same">
    <element-query name="title" ns="http://my/namespace" />
  </constraint>
</options>
```

Further, suppose you submit a `POST /v1/search` request that uses `my-options` and includes the following query options in a combined query in the request body:

```
$ cat body.xml
<search xmlns="http://marklogic.com/appservices/search">
  <options>
    <return-metrics>true</return-metrics>
    <debug>true</debug>
  <constraint name="same">
    <collection prefix="http://server.com/dynamic/"/>
  </constraint>
    <constraint name="different">
      <element-query name="scene" ns="http://my/namespace" />
    </constraint>
  </options>
</search>

# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X POST \
    -d@./body.xml -H "content-type: application/xml" \
    'http://localhost:8003/v1/search?q=TRAGEDY&options=my-options'
```

The query is evaluated with the following merged options. The persistent options contribute the `fragment-scope` option and the constraint named `not-same`. The dynamic options in the combined query contribute the `return-metrics` and `debug` options and the constraints named `same` and `different`. The `return-metrics` setting and the constraint named `same` from `my-options` are discarded.

```
<options xmlns="http://marklogic.com/appservices/search">
  <fragment-scope>properties</fragment-scope>
  <return-metrics>true</return-metrics>
  <debug>true</debug>
  <constraint name="same">
    <collection prefix="http://server.com/dynamic/"/>
  </constraint>
    <constraint name="different">
      <element-query name="scene" ns="http://my/namespace" />
    </constraint>
  </options>
  <constraint name="not-same">
    <element-query name="title" ns="http://my/namespace" />
  </constraint>
</options>
```

4.6.4 Performance Considerations

Using persistent query options usually performs better than using a combined query. In most cases, the difference between the two approaches is slight.

When MarkLogic Server processes a combined query, the per request query options must be parsed and merged with named and default options on every search. When you only use persistent named or default query options, you reduce this overhead.

If your application does not require dynamic per-request query options, you should use the `/config/query/{name}` service to persist your options under a name and use the `options` request parameter to associate the options with a simple string, structured, or values query. For details, see "Configuring Query Options" on page 167.

4.6.5 Combined Query Examples

This section includes the following examples:

- Example: Overriding Persistent Constraints
- Example: Modifying the Search Response

4.6.5.1 Example: Overriding Persistent Constraints

The following example uses a bucketed constraint, backed by an element range index, to group items into facets by price. Assume following options that define price range buckets are stored as persistent options using the `/config/query/{name}` service, as described in "Creating or Modifying Query Options" on page 169.

```
<options xmlns="http://marklogic.com/appservices/search">
 <constraint name="price" facet="true">
    <range type="xs:int">
    <element ns="" name="price"/>
    <bucket name="under50" ge="0" lt="50">under $50</bucket>
    <bucket name="under100" ge="50" lt="101">$50-$100</bucket>
    <bucket name="over100" ge="101">over $100</bucket>
  </range>
 </constraint>
</options>
```

The application can use these persistent options to generate a faceted navigation page that allows users to browse by price. You can use dynamic query options to render a page that includes a custom facet from a price range entered by the user. The resulting combined query might look like the following if the user defined a price range of $100-150:

```
<search xmlns="http://marklogic.com/appservices/search">
  <options>
    <constraint name="price" facet="true">
      <range type="xs:int">
        <element ns="" name="price"/>
        <bucket name="under50" ge="0" lt="50">under $50</bucket>
        <bucket name="under100" ge="50" lt="101">$50-$100</bucket>
        <bucket name="over100" ge="101">over $100</bucket>
        <bucket name="custom" ge="100" lt="151">$100 to $150</bucket>
      </range>
    </constraint>
  </options>
  <query>
    <range-constraint-query>
      <constraint-name>price</constraint-name>
```

```
      <value>custom</value>
    </range-constraint-query>
  <query>
</search>
```

4.6.5.2 Example: Modifying the Search Response

The following example uses a combined query that contains only query options to enable the `return-query` option on the fly to explore how a string query is represented as a `cts:query` after parsing. The `return-query` setting in the dynamic options overrides any `return-query` setting in the default options. All other settings in the default options are unchanged and apply to the query evaluation.

```
$ cat combo-query.xml
<search xmlns="http://marklogic.com/appservices/search">
  <options>
    <return-query>true</return-query>
  </options>
</search>

# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X POST \
    -d@./combo-query.xml -H "content-type: application/xml" \
    'http://localhost:8003/v1/search?q=Horatio NEAR Yorick'

<search:response snippet-format="snippet" total="0" start="1" ...>
  <search:qtext>Horation NEAR Yorick</search:qtext>
  <search:query>
    <cts:near-query qtextjoin="NEAR" strength="30" ...>
      <cts:word-query qtextref="cts:text">
        <cts:text>Horatio</cts:text>
      </cts:word-query>
      <cts:word-query qtextref="cts:text">
        <cts:text>Yorick</cts:text>
      </cts:word-query>
    </cts:near-query>
  </search:query>
  ...
</search:response>
```

For details, see “Interaction with Persistent Query Options” on page 163.

4.7 Querying Triples

You can query semantic data in the database by sending a GET request to the `/graphs/sparql` service with a URL of the following form:

```
http://host:port/version/graphs/sparql?query=sparql-query
```

Optionally, you can define the RDF Dataset over which to query by specifying one or more graph URIs using the `named-graph-uri` and/or `default-graph-uri` request parameters:

```
http://host:port/version/graphs/sparql?query=sparql-query&named-graph-
uri=graph-uri&default-graph-uri=graph-uri
```

You can also specify the dataset within the query. If you specify a dataset in both the request parameters and the query, the dataset defined with `named-graph-uri` and `default-graph-uri` takes precedence. If no dataset is defined in the request parameters or in the query, the dataset includes all triples, regardless of graph.

The SPARQL query in the query request parameter must be URL-encoded.

You can also put the query in the body of a POST request to `/graphs/sparql`. As with the GET request, define the RDF Dataset using `named-graph-uri` and/or `default-graph-uri`. For example, make a POST request with a URL of the following form:

```
http://host:port/version/graphs/sparql?named-graph-uri=graph-uri&defau
lt-graph-uri=graph-uri
```

Note: You must enable the collection lexicon on your database before you can use the semantics REST services or use the GRAPH '?g' construct in a SPARQL query.

For details, see Semantic Queries in *Semantics Developer's Guide*.

4.8 Configuring Query Options

Use the `/config/query` resources to install, list, and manage sets of query options. Use the `options` request parameter to apply installed query options to requests to `/search`, `/keyvalue`, and `/values`. This section covers the following topics:

- Controlling Queries With Options
- Adding Query Options to a Request
- Creating or Modifying Query Options
- Checking Index Availability
- Retrieving Options
- Retrieving a List of Installed Query Options
- Removing Query Options
- Removing All Named Query Options

4.8.1 Controlling Queries With Options

You can use persistent or dynamic query options to customize your queries. Query options provide capabilities such as:

- define word, value and element constraints
- define lexicon and range index specifications
- control search characteristics such as case sensitivity and ordering
- extend the search grammar
- customize query results including pagination, snippeting, and filtering

For details on these and other options, see “Appendix A: JSON Query Options Reference” on page 308 or “Appendix B: XML Query Options Reference” on page 366.

MarkLogic Server comes configured with default query options. You can extend and modify the default options using `/config/query/default`.

Use the `options` request parameter to the `/search`, `/values`, and `/keyvalue` services to customize your queries. Query options provide capabilities such as:

You can also create custom persistent or dynamic query options. Persistent query options are named, pre-defined query options that you install using the `/config/query/{name}`. Dynamic query options are per-request, transient options defined in a combined query. For details, see “Creating or Modifying Query Options” on page 169 and “Specifying Dynamic Query Options with Combined Query” on page 161

Once you install query options under a name, you can apply them to a `/search`, `/keyvalue`, or `/values` request using the `options` request parameter. The following example searches for the word “julius”, using the query options named “my-options”:

```
http://localhost:8003/v1/search?q=julius&options=my-options
```

4.8.2 Adding Query Options to a Request

You can customize a query with query options in the following ways:

- Use the `options` request parameter of a GET request to `/search`, `/keyvalue`, or `/values/{name}` to supply the name of pre-installed persistent options.
- Use the `options` parameter of a POST request to `/search` or `/values/{name}` to supply the name of pre-installed persistent options.
- Use the `options` element (XML) or sub-object (JSON) of a combined query passed in the body of a POST request to `/search` or `/values/{name}` to supply dynamic options.

Pre-installed, persistent query options usually provide better performance. Using dynamic query options introduces option parsing and merging overhead to every query.

Persistent and dynamic query options can be specified in either XML or JSON. Persistent options must be installed before you can use them; for details, see “Creating or Modifying Query Options” on page 169.

Dynamic options are only usable with services that support POSTing a combined query in the request body. Methods that support combined query allow you to specify persistent and dynamic options in the same request. Where both are present, they are merged; in case of a conflict, a dynamic option setting overrides a persistent option setting. For details, see “Specifying Dynamic Query Options with Combined Query” on page 161.

4.8.3 Creating or Modifying Query Options

To install or modify named persistent query options, send a PUT or POST request to the `/config/query` service with a URL of the form:

```
http://host:port/version/config/query/name
```

When constructing the request:

1. Set the *name* portion of the URL to a unique name for these options, or to `default`. Use the name to identify the query options in subsequent request, as described in “Controlling Queries With Options” on page 167.

2. Place the XML or JSON option data in the request body.

   For syntax details, see “Appendix A: JSON Query Options Reference” on page 308 or “Appendix B: XML Query Options Reference” on page 366..

3. Specify the MIME type of the body content in the `format` parameter or the HTTP `Content-type` header, as described in “Controlling Input and Output Content Type” on page 25. You may only send XML or JSON. The default format is XML.

When choosing the HTTP verb, consider the following:

- To install new query options, use either PUT or POST.
- To replace the named query options, use PUT.
- To add options to the named query options, use POST. Any options not included in the payload remain unchanged. New options are added.

If query option validation is enabled, the request will fail if it results in invalid query options. If the request fails, the options are unchanged. Option validation is enabled by default. You can disable option validation using the `validate-options` instance configuration property. For details, see “Configuring Instance Properties” on page 41.

The following example installs query options named “title-only” that define a search constraint named “title”. The constraint limits queries to terms appearing in a `TITLE` element. The query options are then used to find occurrences of “julius” in `TITLE` elements:

```
$ cat bill-options.txt
<options xmlns="http://marklogic.com/appservices/search">
  <constraint name="title">
```

```
    <word>
      <element ns="" name="TITLE" />
    </word>
  </constraint>
</options>

# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -T './bill-options.txt' \
    -H "Content-type: application/xml" \
    http://localhost:8003/v1/config/query/title-only
...
HTTP/1.1 204 Content Updated

$ curl --anyauth --user user:password -X GET \
  'http://localhost:8003/v1/search?q=title:julius&options=title-only'
...
<search:response total="1" start="1" page-length="10"
   xmlns="" xmlns:search="http://marklogic.com/appservices/search">
    ...
</search:response>
```

To add case-sensitivity to the query options installed above, this example sends a POST request to `/config/query`. The body content type, JSON, is given via the `format` request parameter instead of the `Content-type` header.

```
$ cat add-cs.json
{
  "options":
    { "term":
      { "term-option":"case-sensitive" }
    }
}

# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X POST -d@'add-cs.json' \
    -H "Content-type: application/json" \
    http://localhost:8003/v1/config/query/title-only
...
HTTP/1.1 201 Content Created
```

To confirm the change, the modified option is fetched using a GET request:

```
$ curl --anyauth --user user:password -X GET \
    -H "Accept: application/xml" \
    http://localhost:8003/v1/config/query/title-only
...
<options xmlns="http://marklogic.com/appservices/search">

  <constraint name="title">
    <word>
      <element ns="" name="TITLE"/>
    </word>
  </constraint>
```

```
  <term>
    <term-option>case-sensitive</term-option>
  </term>
</options>
```

4.8.4 Creating or Modifying One Option

To add or modify just one setting in a set of query options, send a PUT or POST request to the `/config/query` service with a URL of the form:

```
http://host:port/version/config/query/name/option_name
```

When constructing the request:

1. Set the *name* portion of the URL to the name of the enclosing query options, or to `default`.

2. Set the *option_name* portion of the URL to a query option name, such as `constraint` or `term`.

3. Place the XML or JSON option data in the request body. The data should be an `options` node (XML) or map (JSON) that includes the option named in the URL.

4. Specify the content type of the body in the `format` parameter or the HTTP `Content-type` header, as described in "Controlling Input and Output Content Type" on page 25. You may only send XML or JSON. The default format is XML.

The *option_name* portion of the URL must be the name of an option that can appear as an immediate child of a query options XML node or JSON object. Finer grained access is not supported. For details on query options names and structure, see "Appendix A: JSON Query Options Reference" on page 308 or "Appendix B: XML Query Options Reference" on page 366.

When choosing the HTTP verb, consider the following:

- To add a new option to the set, use either PUT or POST.
- To replace all existing options of the same name, use PUT.
- To add a new occurrence of an existing option that can appear multiple times, use POST. For example, query options can contain multiple `<constraint/>` elements.

If query option validation is enabled, the request will fail if it results in invalid query options. If the request fails, the options are unchanged. Option validation is enabled by default. You can disable option validation using the validate-options instance configuration property. For details, see "Configuring Instance Properties" on page 41.

4.8.5 Checking Index Availability

Some query options require the database configuration to include supporting indexes. For example, if your query options contain a range constraint, then you can only use those options on a database whose configuration includes a corresponding range index.

You can use the /config/indexes service to compare query options to the database configuration and get a report on whether or not all required indexes are present. For missing indexes, the report includes information to help create the missing index. You can either check all query options configurations, or a particular one (by name).

To check all query options configurations, send a GET request to the `/config/indexes` service of the form:

```
http://host:port/version/config/indexes
```

To check a specific query options configuration, send a GET request to the `/config/indexes/{name}` service of the form:

```
http://host:port/version/config/indexes/name
```

Where *name* is the name under which the options were installed using the `/config/query/{name}` service, as described in “Creating or Modifying Query Options” on page 169.

You can request an index report in XML, JSON, or HTML, using either the `format` request parameter or the HTTP Accept headers. The HTML report is a user-friendly report that contains more details.

For example, suppose the following query options are installed under the name “tuples”. These options require 2 range indexes: An element range index on `<SPEAKER/>` and a path range index on the XPath expression `/PLAY/ACT/SCENE/TITLE`:

```
<options xmlns="http://marklogic.com/appservices/search">
  <tuples name="speaker-title">
    <range type="xs:string">
      <element ns="" name="SPEAKER"/>
    </range>
    <range type="xs:string">
      <path-index>/PLAY/ACT/SCENE/TITLE</path-index>
    </range>
  </tuples>
</options>
```

The following command requests an index check check report in XML for the “tuples” options. Use the format parameter or the Accept headers to request a different report format.

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
```

```
-H "Accept: application/xml" \
http://localhost:8003/v1/config/indexes/tuples
```

The table below shows the generated report. Notice that the report indicates that the index configuration is not complete, and shows which query option is not complete. You can use the `path-index` value to create the required path range index.

Format	Example Output

XML	`<rapi:index-summaries xmlns:rapi="http://marklogic.com/rest-api">`
	`  <rapi:index-count>1</rapi:index-count>`
	`  <rapi:complete>false</rapi:complete>`
	`  <rapi:index-summary>`
	`  <rapi:name>`**`/v1/config/query/tuples`**`</rapi:name>`
	`  <rapi:complete>`**`false`**`</rapi:complete>`
	`  <range type="xs:string"`
	`  xmlns="http://marklogic.com/appservices/search">`
	`  <path-index>/PLAY/ACT/SCENE/TITLE</path-index>`
	`  </range>`
	`  </rapi:index-summary>`
	`</rapi:index-summaries>`

JSON	`{`
	`  "index-summaries": {`
	`  "index-summary": [`
	`  {`
	`  "name": "\/v1\/config\/query\/tuples",`
	`  "complete": "false",`
	`  "range": {`
	`  "type": "xs:string",`
	`  "path-index": "\/PLAY\/ACT\/SCENE\/TITLE"`
	`  }`
	`  }`
	`  ],`
	`  "index-count": "1",`
	`  "complete": "false"`
	`  }`
	`}`

4.8.6 Retrieving Options

To retrieve previously installed persistent query options, send a GET request to the `/config/query` service with a URL of the form:

```
http://host:port/version/config/query/name
```

Where name is the *name* of the query options.

MarkLogic Server responds with the contents of the named query options, as XML or JSON. XML is the default format. Use the Accept header or `format` request parameter to select the output content type. For details, see "Controlling Input and Output Content Type" on page 25.

You can also retrieve the settings for a specific option within the query options by sending a GET request with a URL of the form:

```
http://host:port/version/config/query/name/option_name
```

Where *option_name* is an option name. For details, see "Appendix A: JSON Query Options Reference" on page 308 or "Appendix B: XML Query Options Reference" on page 366.

As when retrieving a whole set, results can be requested as XML or JSON. Use the Accept header or `format` request parameter to select the output content type. For details, see "Controlling Input and Output Content Type" on page 25. No version information is returned when examining only one option.

The following example retrieves the contents of the query options called "title-only":

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    http://localhost:8003/v1/config/query/title-only
...
<options xmlns="http://marklogic.com/appservices/search">
  <constraint name="title">
    <word>
      <element ns="" name="TITLE"/>
    </word>
  </constraint>
  <term>
    <term-option>case-sensitive</term-option>
  </term>
</options>
```

The following example retrieves only the term option of the "title-only" query options:

```
$ curl --anyauth --user user:password -X GET \
    http://localhost:8003/v1/config/query/title-only/term
...
<options xmlns="http://marklogic.com/appservices/search">
  <term>
    <term-option>case-sensitive</term-option>
  </term>
</options>
```

4.8.7 Retrieving a List of Installed Query Options

To retrieve a list of the names of all installed query options, send a GET request to `/config/query` with a URL of the form:

```
http://host:port/version/config/query
```

MarkLogic Server responds with a list of names in XML or JSON. XML is the default format. Use the Accept header or `format` request parameter to select the output content type. For details, see "Controlling Input and Output Content Type" on page 25.

If there are no custom query options installed, MarkLogic Server responds with an empty XML `<options>` node or JSON array.

The following example retrieves the list of named query options as XML. The results show 2 sets of query options, named "title-only" and "play-type".

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    http://localhost:8003/v1/config/query
...
<rapi:query-options xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:options>
    <rapi:name>title-only</rapi:name>
    <rapi:uri>/v1/config/query/title-only</rapi:uri>
  </rapi:options>
  <rapi:options>
    <rapi:name>play-type</rapi:name>
    <rapi:uri>/v1/config/query/play-type</rapi:uri>
  </rapi:options>
</rapi:query-options>
```

The following example requests the same information as JSON, using the `format` request parameter:

```
$ curl --anyauth --user user:password -X GET \
    http://localhost:8003/v1/config/query
...
[
  {"name":"title-only","uri":"/v1/config/query/title-only"}
  {"name":"play-type","uri":"/v1/config/query/play-type"}
]
```

4.8.8 Removing Query Options

You can remove a single setting, one set of query options, or all query options, as described in the following topics:

- Removing Query Options
- Removing a Single Option
- Removing All Named Query Options

4.8.8.1 Removing Query Options

To remove the query options installed under a particular name, send a DELETE request to the `/config/query` service with a URL of the form:

```
http://host:port/version/config/query/name
```

Where *name* is the name of the query options to remove. MarkLogic Server responds with 204 if the option set is successfully deleted.

4.8.8.2 Removing a Single Option

To remove a specific setting in a set of query options, send a DELETE request to the `/config/query` service with a URL of the form:

```
http://host:port/version/config/query/name/option_name
```

Where *option_name* is the name of the option to remove and *name* is the name of the containing query options. MarkLogic Server responds with 204 if the option is successfully deleted.

The *option_name* portion of the URL must be the name of an option that can appear as an immediate child of a query options XML node or JSON object. Finer grained access is not supported. All options with this name are removed. For example, if the query options named "my-options" contain multiple `<constraint/>` options, then the following request removes all of them:

```
DELETE http://localhost:8003/v1/config/query/my-options/constraint
```

For details on query options names and structure, see "Appendix A: JSON Query Options Reference" on page 308 or "Appendix B: XML Query Options Reference" on page 366.

If query option validation is enabled, the request will fail if it results in invalid query options. If the request fails, the options are unchanged. Option validation is enabled by default. You can disable option validation using the validate-options instance configuration property. For details, see "Configuring Instance Properties" on page 41.

For details on query option names, see "Appendix A: JSON Query Options Reference" on page 308 or "Appendix B: XML Query Options Reference" on page 366.

4.8.8.3 Removing All Named Query Options

To remove all named query options from a MarkLogic REST API instance, send a DELETE request to the `/config/query` service with a URL of the form:

```
http://host:port/version/config/query
```

MarkLogic Server responds with 204 if the options successfully deleted.

4.9 Using Namespace Bindings

This sections covers the following topics:

- When Do You Need a Namespace Binding
- Creating or Updating a Namespace Binding
- Creating or Updating Multiple Namespace Bindings
- Listing Available Namespace Bindings
- Deleting Namespace Bindings

4.9.1 When Do You Need a Namespace Binding

Use the `/config/namespaces` service to define namespace prefixes that you can use in your queries. For example, to specify an element with QName `theNS:theElement` in a `/keyvalue` search, you might refer to it as follows:

```
http://host:port/version/keyvalue?element=theNS:theElement&value=foo
```

There is no way to dynamically specify the namespace in your request, so you must pre-define the namespace prefix binding using `/config/namespaces` or `/config/namespace/{name}`.

4.9.2 Creating or Updating a Namespace Binding

This section describes how to create a single namespace binding. You can also define multiple bindings in a single request; for details, see “Creating or Updating Multiple Namespace Bindings” on page 179.

To define a namespace binding, send a PUT request to the `/config/namespaces/{name}` service with a URL of the form:

```
http://host:port/version/config/namespaces/name
```

Where *name* is the namespace prefix you want to define. If a binding already exists for this prefix, it is replaced.

When constructing your request:

1. Set the *name* portion of the URL to the desired namespace prefix.

2. Place the XML or JSON binding definition in the request body. See the table below.

3. Specify the content type of the body in the `format` parameter or the HTTP `Content-type` header, as described in “Controlling Input and Output Content Type” on page 25. You may only send XML or JSON. The default format is XML.

The request body must have the following form:

Format	Body
XML	<namespace xmlns="http://marklogic.com/rest-api"> <prefix>*the_prefix*</prefix> <uri>*the_uri*</uri> </namespace>

JSON	{
	  "prefix" : "*the_prefix*",
	  "uri" : "*the_uri*"
	}

The following example binds the prefix “bill” to the namespace URI “http://marklogic.com/examples/shakespeare”:

```
$ cat ns-binding.xml

<namespace xmlns="http://marklogic.com/rest-api">
  <prefix>bill</prefix>
  <uri>http://marklogic.com/examples/shakespeare</uri>
</namespace>

# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT -d@./ns-binding.xml \
    -H "Content-type: application/xml" \
    http://localhost:8004/v1/config/namespaces/bill
```

You can examine the binding by sending a GET request to `/config/namespaces/{name}` or to `/config/namespaces`. The latter lists all bindings. For example:

```
$  curl --anyauth --user user:password -X GET \
    http://localhost:8004/v1/config/namespaces/bill

<rapi:namespace xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:prefix>bill</rapi:prefix>
  <rapi:uri>http://marklogic.com/examples/shakespeare</rapi:uri>
</rapi:namespace>
```

To get the equivalent output as JSON, use the `format` parameter or specify `application/json` in the HTTP Accept header. For example:

```
$  curl --anyauth --user user:password -X GET \
    -H "Accept: application/json" \
    http://localhost:8004/v1/config/namespaces/bill

{
  "prefix":"bill",
  "uri":"http://marklogic.com/examples/shakespeare"
}
```

4.9.3 Creating or Updating Multiple Namespace Bindings

This section describes how to define multiple namespace bindings with a single request. You can also define a specific single binding; for details, see “Creating or Updating a Namespace Binding” on page 177.

To define multiple namespace bindings, send a PUT or POST request to the `/config/namespaces` service with a URL of the form:

```
http://host:port/version/config/namespaces
```

When constructing your request:

1. Choose PUT to replace all bindings with those the request body. Choose POST to append to existing bindings.

2. Place the XML or JSON binding definitions in the request body. See the table below.

3. Specify the content type of the body in the `format` parameter or the HTTP `Content-type` header, as described in “Controlling Input and Output Content Type” on page 25. You may only send XML or JSON. The default format is XML.

If you use POST and a binding already exists for one defined in the request body, MarkLogic Server returns status 400 (Bad Request).

The request body must have the following form:

Format	Body
XML	<namespace-bindings xmlns="http://marklogic.com/rest-api"> <namespace> <prefix>*a_prefix*</prefix> <uri>*a_namespace_uri*</uri> </namespace> <namespace> <prefix>*another_prefix*</prefix> <uri>*another_namespace_uri*</uri> </namespace> </namespace-bindings>

JSON	{
	  "namespace-bindings": [
	  {
	  "prefix": "*a_prefix*",
	  "uri": "*a_namespace_uri*"
	  },
	  {
	  "prefix": "*another_prefix*",
	  "uri": "*another_namespace_uri*"
	  }
	  ]
	}

The following example binds the prefix “one” to the namespace URI “http://marklogic.com/examples/one” and the prefix “two” to the namespace URI “http://marklogic.com/examples/two”:

```
$ cat ns-bindings.xml
<namespace-bindings xmlns="http://marklogic.com/rest-api">
  <namespace>
    <prefix>one</prefix>
    <uri>http://marklogic.com/examples/one</uri>
  </namespace>
  <namespace>
    <prefix>two</prefix>
    <uri>http://marklogic.com/examples/two</uri>
  </namespace>
</namespace-bindings>

# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT -d@./ns-bindings.xml \
    -H "Content-type: application/xml" \
    http://localhost:8004/v1/config/namespaces
```

The following example creates equivalent bindings using JSON input:

```
$ cat ns-bindings.json
{
  "namespace-bindings": [
    {
      "prefix": "one",
      "uri": "http:\/\/marklogic.com\/examples\/one"
    },
    {
      "prefix": "two",
      "uri": "http:\/\/marklogic.com\/examples\/two"
    }
  ]
}

$ curl --anyauth --user user:password -X PUT -d@./ns-bindings.json \
    -H "Content-type: application/json" \
    http://localhost:8004/v1/config/namespaces
```

You can examine the binding by sending a GET request to `/config/namespaces/{name}` or to `/config/namespaces`.

4.9.4 Listing Available Namespace Bindings

To list all available namespace bindings, send a GET request to /config/namespaces with a URL of the form:

```
http://host:port/version/config/namespaces
```

To retrieve the binding for a single namespace prefix, send a GET request to /config/namespaces/{name} with a URL of the form:

```
http://host:port/version/config/namespaces/name
```

Where *name* is a bound namespace prefix.

You can request output as either XML or JSON. Use the Accept header or `format` request parameter to select the output content type. For details, see “Controlling Input and Output Content Type” on page 25.

The following example command requests all namespace bindings, as XML:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    -H "Accept: application/xml" \
    http://localhost:8004/v1/config/namespaces
```

The output from the command is shown in the table below, assuming two namespace bindings are installed, for the prefixes “one” and “two”.

Format	Example Output
XML	<rapi:namespace-bindings xmlns:rapi="http://marklogic.com/rest-api"> <rapi:namespace> <rapi:prefix>one</rapi:prefix> <rapi:uri>http://marklogic.com/examples/one</rapi:uri> </rapi:namespace> <rapi:namespace> <rapi:prefix>two</rapi:prefix> <rapi:uri>http://marklogic.com/examples/two</rapi:uri> </rapi:namespace> </rapi:namespace-bindings>

JSON	{
	"namespace-bindings": [
	{
	"prefix": "one",
	"uri": "http://marklogic.com/examples/one"
	},
	{
	"prefix": "two",
	"uri": "http://marklogic.com/examples/two"
	}
	]
	}

When you use `GET /config/namespaces/{name}`, the output is similar, but without the namespace-bindings wrapper. For example to retrieve the binding for the “one” prefix:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    -H "Accept: application/xml" \
    http://localhost:8004/v1/config/namespaces/one

<rapi:namespace xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:prefix>one</rapi:prefix>
  <rapi:uri>http://marklogic.com/examples/one</rapi:uri>
</rapi:namespace>
```

The following command retrieves the same information as JSON:

```
$ curl --anyauth --user user:password -X GET \
    -H "Accept: application/json" \
    http://localhost:8004/v1/config/namespaces/one

{"prefix":"one","uri":"http://marklogic.com/examples/one"}
```

4.9.5 Deleting Namespace Bindings

To remove all namespace bindings, send a DELETE request to /config/namespaces with a URL of the form:

```
http://host:port/version/config/namespaces
```

To remove a specific binding, send a DELETE request to /config/namespaces/{name} with a URL of the form:

```
http://host:port/version/config/namespaces/name
```

Where *name* is a bound namespace prefix. If no binding exists for name, MarkLogic Server returns status 404 (Not Found).

The following example deletes just the binding for the prefix “one”:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X DELETE \
    http://localhost:8004/v1/config/namespaces/one
```

4.10 Generating Search Facets

The MarkLogic Server Search API enables you to expose search facets in your application. Facets enable users to filter search results by narrowing down the search criteria. To learn more about facets, see Constrained Searches and Faceted Navigation in the *Search Developer’s Guide*.

To generate facet information in search results, use query options that include constraints that support facets, such as collection and element constraints. You can also define custom constraints; see Creating a Custom Constraint in the *Search Developer’s Guide*.

The following example returns facet information about play types and enables searching by the facet “type”, assuming the documents in the database have been added to the collections `/play-type/Comedy`, `/play-type/Tragedy`, and `/play-type/History`. For more examples, see Constraint Options in the *Search Developer’s Guide*.

> **Note:** This example uses a collection constraint. To search on a collection facet, you must enable the “collection lexicon” database configuration parameter in the Admin Interface.

To enable the play type facet, create query options that define a collection constraint:

Format	Query Options
XML	<options xmlns="http://marklogic.com/appservices/search"> <constraint name="type"> <collection prefix="/play-type/"/> </constraint> <return-facets>true</return-facets> </options>

JSON	{
	  "options": {
	  "constraint": [
	  {
	  "name": "type",
	  "collection": { "prefix": "\/play-type\/" }
	  }],
	  "return-facets": true
	  }
	}

Install the options with the name “facet-options”:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT \
    -d@"./facet-options.xml" -H "Content-type: application/xml" \
    http://localhost:8003/v1/config/query/facet-options
```

If you query `/search` or `/keyvalue` using “index-options” as the query options, facet results are included in the search results. For example, the following query finds all occurrences of “castle” and the search response includes a count of the matches for each play type:

```
$ curl --anyauth --user user:password -X GET \
  'http://localhost:8003/v1/search?options=facet-options&q=castle'
...
<search:response snippet-format="snippet" total="88" ...>
  <search:result />
  <search:facet name="type" type="collection">
    <search:facet-value name="Comedy" count="5">
      Comedy
    </search:facet-value>
    <search:facet-value name="History" count="35">
      History
    </search:facet-value>
    <search:facet-value name="Tragedy" count="47">
      Tragedy
    </search:facet-value>
  </search:facet>
...
</search:response>
```

To get the equivalent results as JSON, use `format=json` or include `application/json` in the HTTP Accept header. For example:

```
$ curl --anyauth --user user:password -X GET \
  -H "Accept: application/json" \
  'http://localhost:8003/v1/search?options=facet-options&q=castle'
...
{
  "snippet-format": "snippet",
  "total": 88,
  "start": 1,
  "page-length": 10,
  "results": [...],
  "facets": {
    "type": {
      "type": "collection",
      "facetValues": [
        {
          "name": "Comedy",
          "count": 5
        },
        {
          "name": "History",
          "count": 35
        },
        {
          "name": "Tragedy",
          "count": 47
        }
      ]
    }
  },
  "qtext": "castle",
  ...
}
```

You can query by facet by including a `facet-name:facet-value` search term. The following command matches occurrences of “castle” in plays of type Comedy:

```
$ curl --anyauth --user user:password -X GET \
    'http://localhost:8003/v1/search?options=facet-options&q=castle
type:Comedy'
```

4.11 Paginating Results

When you query the database using `/search` and `/keyvalues`, you can paginate the query results using the `start` and `pageLength` request parameters. Use `start` to specify the index of the first result to return, and `pageLength` to control the number results to return.

By default, queries return the first 10 results. That is, the default start position is 1 and the default page length is 10. You can fetch successive non-overlapping pages of results by incrementing the start position by the page length in each call.

For more information, see the Search Developer's Guide and Fast Pagination and Unfiltered Searches in the *Scalability, Availability, and Failover Guide*.

The following example command fetches the first 5 results matching "castle". Notice that the search response includes the total number of matches and each `search:result` includes an index.

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    'http://localhost:8003/v1/search?q=castle&start=1&pageLength=5'
...
<search:response snippet-format="snippet" total="88"
    start="1" page-length="5" ...>
  <search:result index="1" uri="/shakespeare/plays/hen_vi_2.xml" .../>
  <search:result index="2" uri="/shakespeare/plays/rich_ii.xml" .../>
  <search:result index="3" uri="/shakespeare/plays/macbeth.xml" .../>
  <search:result index="4" uri="/shakespeare/plays/hen_vi_3.xml" .../>
  <search:result index="5" uri="/shakespeare/plays/othello.xml" .../>
...
</search:response>
```

To fetch the next 5 results, increment start by 5:

```
$ curl --anyauth --user user:password -X GET \
    'http://localhost:8003/v1/search?q=castle&start=6&pageLength=5'
...
<search:response snippet-format="snippet" total="88"
    start="1" page-length="5" ...>
  <search:result index="6" uri="/shakespeare/plays/lear.xml" .../>
  <search:result index="7" uri="/shakespeare/plays/hamlet.xml" .../>
  <search:result index="8" uri="/shakespeare/plays/rich_iii.xml" .../>
  <search:result index="9" uri="/shakespeare/plays/hen_v.xml" .../>
  <search:result index="10" uri="/shakespeare/plays/m_wives.xml" .../>
...
</search:response>
```

4.12 Customizing Search Results

This section covers several features that the MarkLogic REST API provides for search result customization.

- The `transform-results` query option enables you to fine tune the default snippets. For example, you can control how many matches to return.
- Custom snippet extensions identified in query options enable you to modify the contents of snippets returned in the `search:match` portion of a `search:response`.
- Tranform functions enable you to completely change the structure returned by a search or values query. Your transform takes a search:response as input and produces a result document.

The following topics cover these features:

- Customizing Search Snippets
- Transforming the Search Response

4.12.1 Customizing Search Snippets

Search results usually include portions of matching documents with the search matches highlighted, perhaps with some text showing the context of the search matches. These search result pieces are known as *snippets*. MarkLogic Server has a default search snippet format, but you can customize the snippet format by either modifying the configuration of the default snippet function or creating a custom snippet extension.

This section covers the following topics:

- Customizing the Default Search Snippets
- Creating Your Own Snippet Extension
- Generating Custom JSON Snippets

4.12.1.1 Customizing the Default Search Snippets

MarkLogic Server creates snippets by applying a default transformation to search results. You can modify the results of the default transformation using the `transform-results` query option, as described in Modifying Your Snippet Results in the *Search Developer's Guide*. To use this feature with the REST API:

1. Create query options that contain a `transform-results` XML element or JSON object.

2. Install these query options under a name or as the default options using the `/config/query` service, or include them directly in a combined query. For details, see "Configuring Query Options" on page 167.

3. If you installed the configuration as named query options, apply the options to your query by supplying the name in the `options` request parameter. For details, see "Controlling Queries With Options" on page 167.

4. If you installed the configuration as default options, they will be automatically applied to search results returned by any query that does not use named query options.

The following example searches for the term "hamlet" in a collection of Shakespeare plays using the default snippet options. The search results include four snippets in each match.

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    -H "Accept: application/xml" \
    http://localhost:8003/v1/search?q=hamlet
```

```
<search:response snippet-format="snippet" total="20" ...>
  <search:result index="1" uri="/shakespeare/plays/hamlet.xml" ...>
    <search:snippet>
      <search:match
        path="fn:doc(&quot;/shakespeare/plays/hamlet.xml&quot;)/PLAY/T
ITLE">The Tragedy of <search:highlight>Hamlet</search:highlight>,
Prince of Denmark
      </search:match>
      <search:match
        path="fn:doc(&quot;/shakespeare/plays/hamlet.xml&quot;)/PLAY/P
ERSONAE/PERSONA[2]"><search:highlight>HAMLET</search:highlight>, son
to the late, and nephew to the present king.
      </search:match>
      <search:match
        path="fn:doc(&quot;/shakespeare/plays/hamlet.xml&quot;)/PLAY/P
ERSONAE/PERSONA[4]">HORATIO, friend to
<search:highlight>Hamlet</search:highlight>.
      </search:match>
      <search:match
        path="fn:doc(&quot;/shakespeare/plays/hamlet.xml&quot;)/PLAY/P
ERSONAE/PERSONA[16]">GERTRUDE, queen of Denmark, and mother to
<search:highlight>Hamlet</search:highlight>.
      </search:match>
    </search:snippet>
  </search:result>
...
</search:response>
```

Notice that the resulting snippets are from the `<TITLE/>` and `<PERSONAE/>` elements. If you apply the options below to the search instead, at most three snippets are returned and only matches in `<LINE/>` elements are included.

Format	Query Options

XML	`<options xmlns="http://marklogic.com/appservices/search">`
	`  <transform-results apply="snippet">`
	`  <max-matches>3</max-matches>`
	`  <preferred-elements>`
	`  <element ns="" name="LINE" />`
	`  </preferred-elements>`
	`  </transform-results>`
	`  <search-option>filtered</search-option>`
	`</options>`

Format	Query Options

JSON	<pre>{
	  "options": {
	    "transform-results": {
	      "apply": "snippet",
	      "max-matches": 3,
	      "preferred-elements": [
	        {
	          "ns": "",
	          "name": "LINE"
	        }
	      ]
	    },
	    "search-option": true
	  }
	}</pre>

This example installs the above XML options and applies them to the same query:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT \
    -d @./transform-results.xml \
    -H "Content-type: application/xml" \
    http://localhost:8003/v1/config/query/snippet-lines

$ curl --anyauth --user user:password -X GET \
    -H "Accept: application/xml" \
    "http://localhost:8003/v1/search?q=hamlet&options=snippet-lines"

<search:response snippet-format="snippet" total="20" ...>
  <search:result index="1" uri="/shakespeare/plays/hamlet.xml" ...>
    <search:snippet>
      <search:match
        path="fn:doc(&quot;/shakespeare/plays/hamlet.xml&quot;)/PLAY/A
CT[1]/SCENE[1]/SPEECH[48]/LINE[6]">Dared to the combat; in which our
valiant <search:highlight>Hamlet</search:highlight>--
      </search:match>
      <search:match
        path="fn:doc(&quot;/shakespeare/plays/hamlet.xml&quot;)/PLAY/A
CT[1]/SCENE[1]/SPEECH[48]/LINE[17]">His fell to
<search:highlight>Hamlet</search:highlight>. Now, sir, young
Fortinbras,
      </search:match>
      <search:match
        path="fn:doc(&quot;/shakespeare/plays/hamlet.xml&quot;)/PLAY/A
CT[1]/SCENE[1]/SPEECH[59]/LINE[6]">Unto young
<search:highlight>Hamlet</search:highlight>; for, upon my life,
      </search:match>
    </search:snippet>
  </search:result>
...
</search:response>
```

To install the equivalent JSON options, set the Content-type header to `application/json`. For example:

```
$ curl --anyauth --user user:password -X PUT \
    -d @./transform-results.json \
    -H "Content-type: application/json" \
    http://localhost:8003/v1/config/query/snippet-lines
```

To receive JSON search results, set the Accept header to `application/json` or use the `format` request parameter. For example:

```
$ curl --anyauth --user user:password -X GET \
    -H "Accept: application/json" \
    "http://localhost:8003/v1/search?q=hamlet&options=snippet-lines"

{
  "snippet-format": "snippet",
  "total": 20,
  "start": 1,
  "page-length": 10,
  "results": [
    {
      "index": 1,
      "uri": "\/shakespeare\/plays\/hamlet.xml",
      "path": "fn:doc(\"\/shakespeare\/plays\/hamlet.xml\")",
      "score": 158720,
      "confidence": 0.80079,
      "fitness": 1,
      "matches": [
        {
          "path":
"fn:doc(\"\/shakespeare\/plays\/hamlet.xml\")\/PLAY\/ACT[1]\/SCENE[1]\
/SPEECH[48]\/LINE[6]",
          "match-text": [
            "Dared to the combat; in which our valiant ",
            {
              "highlight": "Hamlet"
            },
            "--"
          ]
        },
        {
          "path":
"fn:doc(\"\/shakespeare\/plays\/hamlet.xml\")\/PLAY\/ACT[1]\/SCENE[1]\
/SPEECH[48]\/LINE[17]",
          "match-text": [
            "His fell to ",
            {
              "highlight": "Hamlet"
            },
            ". Now, sir, young Fortinbras,"
          ]
        },
```

```
        {
          "path":
"fn:doc(\"\/shakespeare\/plays\/hamlet.xml\")\/PLAY\/ACT[1]\/SCENE[1]\
/SPEECH[59]\/LINE[6]",
          "match-text": [
            "Unto young ",
            {
              "highlight": "Hamlet"
            },
            "; for, upon my life,"
          ]
        }
      ]
    }
  ],
  ...
}
```

4.12.1.2 Creating Your Own Snippet Extension

If the `tranform-results` query options with the default snippet format does not meet the needs of your application, you can create a custom snippet transformation function, as described in Specifying Your Own Code in transform-results in the *Search Developer's Guide*.

Install custom snippet transformations using the `/ext` service to load them into the Modules database associated with your REST API instance. Use the `apply`, `at`, and `ns` attributes of the `transform-results` query option to specify your custom module.

To create and use a custom snippeting function:

1. Create an XQuery library module that implements your custom snippet function, as described in Specifying Your Own Code in transform-results in the *Search Developer's Guide*.

2. Install your module in the modules database of your REST API instance using the `/ext` service, similar to installing a dependent library for a resource service extension. For details, see "Installing or Updating an Asset" on page 300.

3. Create and install query options that include a `transform-results` option that uses your custom snippeting function. For details, see Search Customization Via Options and Extensions in the *Search Developer's Guide*.

4. Apply the options to your query by supplying the name of the options with the `options` request parameter. For details, see "Controlling Queries With Options" on page 167.

If your application requires non-trivial JSON snippet customization, your snippet function must generate the XML representation of the desired JSON. For details, see "Generating Custom JSON Snippets" on page 196.

The following example builds on the example from "Customizing the Default Search Snippets" on page 187. The example searches for the term "hamlet" in a collection of Shakespeare plays using the default snippet format with the custom options that extract the first three matches in lines of dialog. These options are installed under the name "snippet-lines". With the default snippet transformation function, the search produces the following results:

```
# Install the options
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT \
    -d @./transform-results.xml \
    -H "Content-type: application/xml" \
    http://localhost:8003/v1/config/query/snippet-lines

# Use the options in a query
$ curl --anyauth --user user:password -X GET \
    -H "Accept: application/xml" \
    "http://localhost:8003/v1/search?q=hamlet&options=snippet-lines"

<search:response snippet-format="snippet" total="20" ...>
  <search:result index="1" uri="/shakespeare/plays/hamlet.xml" ...>
    <search:snippet>
      <search:match
        path="fn:doc(&quot;/shakespeare/plays/hamlet.xml&quot;)/PLAY/A
CT[1]/SCENE[1]/SPEECH[48]/LINE[6]">Dared to the combat; in which our
valiant <search:highlight>Hamlet</search:highlight>--
      </search:match>
      <search:match
        path="fn:doc(&quot;/shakespeare/plays/hamlet.xml&quot;)/PLAY/A
CT[1]/SCENE[1]/SPEECH[48]/LINE[17]">His fell to
<search:highlight>Hamlet</search:highlight>. Now, sir, young
Fortinbras,
      </search:match>
      <search:match
        path="fn:doc(&quot;/shakespeare/plays/hamlet.xml&quot;)/PLAY/A
CT[1]/SCENE[1]/SPEECH[59]/LINE[6]">Unto young
<search:highlight>Hamlet</search:highlight>; for, upon my life,
      </search:match>
    </search:snippet>
  </search:result>
...
</search:response>
```

The custom snippeting function below returns the act, scene, speech and line number in each match, instead of the text surrounding the search term.

```
xquery version "1.0-ml";

module namespace example = "http://marklogic.com/example";

import module namespace search =
  "http://marklogic.com/appservices/search"
  at "/MarkLogic/appservices/search/search.xqy";
```

```
declare function example:snippet(
   $result as node(),
   $ctsquery as schema-element(cts:query),
   $options as element(search:transform-results)?
) as element(search:snippet)
{
  let $default-snippet := search:snippet($result, $ctsquery, $options)
  return element
    { fn:QName(fn:namespace-uri($default-snippet),
               fn:name($default-snippet)) }
    { $default-snippet/@*,
      for $child in $default-snippet/node()
      return
        if ($child instance of element(search:match))
        then element {fn:QName(fn:namespace-uri($child),
                     fn:name($child)) } {
          $child/@*,
          let $parts :=
            fn:tokenize(
              fn:substring-after($child/@path, "/PLAY/ACT["), "\[")
          let $location := fn:concat(
            "Act ", fn:substring-before($parts[1], "]"),
            ", Scene ", fn:substring-before($parts[2], "]"),
            ", Speech ", fn:substring-before($parts[3], "]"),
            ", Line ", fn:substring-before($parts[4], "]"))
          return text {$location}
        }
        else $child
    }
};
```

If the above module is installed in the instance modules database with the URI `/ext/my.domain/my-snippets.xqy`, then we can use the following options to generate custom snippets by supplying the localname of the custom function (`snippet`) in `apply`, the namespace of the module (`http://marklogic.com/example`) in `ns`, and the URI of the module in the instance modules database (`/my.domain/my-snippets.xqy`) in `at`.

Format	Query Options

XML	`<options xmlns="http://marklogic.com/appservices/search">`
	`  <transform-results apply="snippet"`
	`  ns="http://marklogic.com/example"`
	`  at="/ext/my.domain/my-snippets.xqy">`
	`  <max-matches>3</max-matches>`
	`  <preferred-elements>`
	`  <element ns="" name="LINE" />`
	`  </preferred-elements>`
	`  </transform-results>`
	`  <search-option>filtered</search-option>`
	`</options>`

JSON	`{`
	`  "options": {`
	`  "transform-results": {`
	`  "apply": "snippet",`
	`  "ns": "http://marklogic.com/example",`
	`  "at": "/ext/my.domain/my-snippets.xqy",`
	`  "max-matches": 3,`
	`  "preferred-elements": [`
	`  {`
	`  "ns": "",`
	`  "name": "LINE"`
	`  }`
	`  ]`
	`  },`
	`  "search-option": true`
	`  }`
	`}`

This example installs the above XML options and applies them to the query for “hamlet” to produce snippets that display the act, scene, speech and dialog number of each match:

```
# Windows users, see Modifying the Example Commands for Windows

# Install the custom snippeting module
$ curl --anyauth --user user:password -X PUT \
    -d @./my-snippets.xqy \
    -H "Content-type: application/xquery" \
    http://localhost:8003/v1/ext/my.domain/my-snippets.xqy
```

```
# Install the options
$ curl --anyauth --user user:password -X PUT \
    -d @./transform-results.xml \
    -H "Content-type: application/xml" \
    http://localhost:8003/v1/config/query/custom-snippet

# Use the options in a query
$ curl --anyauth --user user:password -X GET \
    -H "Accept: application/xml" \
    "http://localhost:8003/v1/search?q=hamlet&options=custom-snippet"

<search:response snippet-format="custom" total="20" start="1" ...>
  <search:result index="1" uri="/shakespeare/plays/hamlet.xml"
      path="fn:doc(&quot;/shakespeare/plays/hamlet.xml&quot;)" ...>
    <search:snippet>
      <search:match
        path="fn:doc(&quot;/shakespeare/plays/hamlet.xml&quot;)/PLAY/A
CT[1]/SCENE[1]/SPEECH[48]/LINE[6]">Act 1, Scene 1, Speech 48, Line 6
      </search:match>
      <search:match
        path="fn:doc(&quot;/shakespeare/plays/hamlet.xml&quot;)/PLAY/A
CT[1]/SCENE[1]/SPEECH[48]/LINE[17]">Act 1, Scene 1, Speech 48, Line 17
      </search:match>
      <search:match
        path="fn:doc(&quot;/shakespeare/plays/hamlet.xml&quot;)/PLAY/A
CT[1]/SCENE[1]/SPEECH[59]/LINE[6]">Act 1, Scene 1, Speech 59, Line 6
      </search:match>
    </search:snippet>
  </search:result>
  ...
</search:response>
```

To generate the equivalent results in JSON, change the Accept header to `application/xml` or use the `format` request parameter. For example:

```
$ curl --anyauth --user user:password -X GET \
    -H "Accept: application/json" \
    "http://localhost:8003/v1/search?q=hamlet&options=custom-snippet"

{
  "snippet-format": "custom",
  "total": 20,
  "start": 1,
  "page-length": 10,
  "results": [
    {
      "index": 1,
      "uri": "\/shakespeare\/plays\/hamlet.xml",
      "path": "fn:doc(\"\/shakespeare\/plays\/hamlet.xml\")",
      "score": 158720,
      "confidence": 0.80079,
      "fitness": 1,
      "matches": [
```

```
        {
          "path":
"fn:doc(\"\/shakespeare\/plays\/hamlet.xml\")\/PLAY\/ACT[1]\/SCENE[1]\
/SPEECH[48]\/LINE[6]",
          "match-text": [
            "Act 1, Scene 1, Speech 48, Line 6"
          ]
        },
        {
          "path":
"fn:doc(\"\/shakespeare\/plays\/hamlet.xml\")\/PLAY\/ACT[1]\/SCENE[1]\
/SPEECH[48]\/LINE[17]",
          "match-text": [
            "Act 1, Scene 1, Speech 48, Line 17"
          ]
        },
        {
          "path":
"fn:doc(\"\/shakespeare\/plays\/hamlet.xml\")\/PLAY\/ACT[1]\/SCENE[1]\
/SPEECH[59]\/LINE[6]",
          "match-text": [
            "Act 1, Scene 1, Speech 59, Line 6"
          ]
        }
      ]
    }
  ],
  ...
}
```

4.12.1.3 Generating Custom JSON Snippets

To include JSON objects and arrays in your snippets, create a custom snippet function that produces the equivalent XML representation in the namespace `http://marklogic.com/xdmp/json/basic`. MarkLogic Server converts conforming XML to JSON when it generates a JSON response. For information on how to represent JSON as XML for successful conversion, see “Working With JSON” on page 206 in *Application Developer’s Guide*.

Recall that the example from “Creating Your Own Snippet Extension” on page 191 changes the match text to display the act, scene, speech, and line number for each match. Since the customization produces simply a text node, it renders automatically in JSON as follows:

```
"match-text": [
  "Act 1, Scene 1, Speech 59, Line 6"
]
```

To return a key-value pair containing the match location information instead, you must explicitly construct XML that can be rendered as JSON. For example, suppose you want to return the following for `match-text` instead of an array of strings:

```
"match-text": { "location": "Act 1, Scene 1, Speech 59, Line 6" }
```

Then your snippet function should generate XML equivalent to the following:

```
<json xmlns="http://marklogic.com/xdmp/json/basic" type="object">
  <location type="string">Act 1, Scene 1, Speech 59, Line 6</location>
</json>
```

The following modified snippet function produces output of the above form. The portions of the code that were modified from the example in "Creating Your Own Snippet Extension" on page 191 are shown in **bold**. The query options and query remain the same.

```
xquery version "1.0-ml";

module namespace example = "http://marklogic.com/example";

import module namespace search =
  "http://marklogic.com/appservices/search"
  at "/MarkLogic/appservices/search/search.xqy";
import module namespace json = "http://marklogic.com/xdmp/json"
    at "/MarkLogic/json/json.xqy";


declare function example:snippet(
   $result as node(),
   $ctsquery as schema-element(cts:query),
   $options as element(search:transform-results)?
) as element(search:snippet)
{
  let $default-snippet := search:snippet($result, $ctsquery, $options)
  return element
    { fn:QName(fn:namespace-uri($default-snippet),
               fn:name($default-snippet)) }
    { $default-snippet/@*,
      for $child in $default-snippet/node()
      return
        if ($child instance of element(search:match))
        then element {fn:QName(fn:namespace-uri($child),
                      fn:name($child)) } {
          $child/@*,
          let $parts :=
            fn:tokenize(
              fn:substring-after($child/@path, "/PLAY/ACT["), "\[")
          let $location := fn:concat(
            "Act ", fn:substring-before($parts[1], "]"),
            ", Scene ", fn:substring-before($parts[2], "]"),
            ", Speech ", fn:substring-before($parts[3], "]"),
            ", Line ", fn:substring-before($parts[4], "]"))
          return element
            {fn:QName("http://marklogic.com/xdmp/json/basic", "json")}
            {
              attribute type {"object"},
              element
                {fn:QName("http://marklogic.com/xdmp/json/basic",
                          "location")}
```

```
                    {
                      attribute type {"string"},
                      text {$location}
                    }
                }
            }
            else $child
        }
    };
```

4.12.2 Transforming the Search Response

You can make arbitrary changes to the response from a search or values query by applying a transformation function to the response.

Search response transforms use the same interface and framework as content transformations applied during document ingestion, described in “Working With Content Transformations” on page 269.

As input, your transform function receives the raw XML search result data prepared by MarkLogic Server. For example, a document with a `<search:response/>` root element. Any customizations made by the `transform-results` query option or result decorators are applied before calling your transform function.

If your function produces XML output and the client application requested JSON output, MarkLogic Server automatically transforms the output of your function to JSON if one of the following conditions are met:

- Your function produces an XML document that conforms to the “normal” output from the search operation. For example, a document with a `<search:response/>` root element whose contents are changed in a way that preserves the usual response structure.
- Your function produces an XML document with a root element in the namespace `http://marklogic.com/xdmp/json/basic` that can be transformed by `json:transform-to-json`.

Under all other circumstances, the output returned by your transform function is returned in the response body as-is.

To use a transform function:

1. Create a transform function according to the interface described in “Writing Transformations” on page 269.

2. Install your transform function on the REST API instance following the instructions in “Installing Transformations” on page 277.

3. Apply your transform to a request to the `/search` or `/values` service by specifying the name in the `transform` request parameter, as described in "Applying Transformations" on page 278.

For example, assuming you installed a transform function under the name "example", the following command applies the function to the results of a search:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
   'http://localhost:8003/v1/search?q=dog&transform=example'
```

4.13 Generating Search Term Completion Suggestions

Use the `/suggest` service to generate search term completion suggestions that match a wildcard terminated string. For example, if the user enters the text "doc" into a search box, you can query `/suggest` with "doc" to retrieve a list of terms matching "doc*", and then display them to user. This service is analogous to calling the XQuery function `search:suggest`.

The following topics are covered:

- Basic Steps
- Example: Generating Search Suggestions Using GET
- Example: Generating Search Suggestions Using POST
- Where to Find More Information

4.13.1 Basic Steps

To retrieve a list of search suggestions using the MarkLogic REST API, use the following procedure. For a detailed example, see "Example: Generating Search Suggestions Using GET" on page 200.

1. Configure at least one index on the XML element, XML attribute, or JSON key values you want to include in the search for suggestions. For performance reasons, a range or collection index is recommended over a word lexicon; for details, see `search:suggest`.

2. Define query options that use your index as a suggestion source by including it in the definition of a `default-suggestion-source` or `suggestion-source option`. For details, see Search Term Completion Using search:suggest in the *Search Developer's Guide*.

3. Send a GET or POST request to the `/suggest` service with a URL of the following form, where `partial-q` is the text for which to retrieve suggestions.

   ```
   http://host:port/version/suggest?partial-q=text_to_match&options=your_
   options
   ```

If you use a GET request to generate suggestions, you must pre-install the query options. For details, see “Configuring Query Options” on page 167.

If you use a POST request to generate suggestions, the POST body must contains a combined query. You can use persistent query options and/or include dynamic query options in the combined query. For details, see “Specifying Dynamic Query Options with Combined Query” on page 161.

If you define your suggestion source in the default query options, you can omit the `options` request parameter or dynamic query options.

You can further constrain the suggestions by including one or more additional queries using the `q` request parameter or in the combined query of a POST request body. The request returns only suggestions in documents that also match the additional queries. Multiple additional queries are AND’d together. The following example URL returns only those suggestions found in fragments that match the query `prefix:xdmp`.

```
http://localhost:8003/v1/suggest/partial-q=doc&q=prefix:xdmp
```

Additional request parameters allow you to limit the number of suggestions returned and specify a substring of `partial-q` to match against. For details, see `GET /v1/suggest` or `POST /v1/suggest` in *MarkLogic REST API Reference*.

4.13.2 Example: Generating Search Suggestions Using GET

This example walks you through configuring your database and REST instance to try retrieving search suggestions. The Documents database is assumed in this example, but you can use any database.

1. If you do not already have a REST API instance, create one. This example assumes your instance is on port 8003.

2. Initialize the Database.

3. Install Query Options

4. Retrieve Unconstrained Search Suggestions

5. Retrieve Constrained Search Suggestions

4.13.2.1 Initialize the Database

Run the following query in Query Console to load the sample data into your database, or use PUT or POST requests to `/documents` to create equivalent documents. The example will retrieve suggestions for the `<name/>` element, with and without a constraint based on the `<prefix/>` element.

```
xdmp:document-insert("/suggest/load.xml",
  <function>
    <prefix>xdmp</prefix>
    <name>document-load</name>
  </function>
  );
xdmp:document-insert("/suggest/insert.xml",
  <function>
    <prefix>xdmp</prefix>
    <name>document-insert</name>
  </function>
  );
xdmp:document-insert("/suggest/query.xml",
  <function>
    <prefix>cts</prefix>
    <name>document-query</name>
  </function>
  );
xdmp:document-insert("/suggest/search.xml",
  <function>
    <prefix>cts</prefix>
    <name>search</name>
  </function>
  );
```

To create the range index used by the example, run the following query in Query Console, or use the Admin Interface to create an equivalent index on the `name` element. The following query assumes you are using the Documents database; modify as needed.

```
xquery version "1.0-ml";
import module namespace admin = "http://marklogic.com/xdmp/admin"
  at "/MarkLogic/admin.xqy";
admin:save-configuration(
  admin:database-add-range-element-index(
    admin:get-configuration(),
    xdmp:database("Documents"),
    admin:database-range-element-index(
    "string", "http://marklogic.com/example",
    "name", "http://marklogic.com/collation/", fn:false())
  )
);
```

4.13.2.2 Install Query Options

This step installs persistent query options that define `name` as the default suggestion source, plus an additional value constraint on `prefix`. Copy the following query options into a file named `suggest-options.xml`:

```
<options xmlns="http://marklogic.com/appservices/search">
 <default-suggestion-source>
   <range type="xs:string" facet="true">
      <element ns="http://marklogic.com/example" name="name"/>
   </range>
 </default-suggestion-source>
 <constraint name="prefix">
   <value>
      <element ns="http://marklogic.com/example" name="prefix"/>
   </value>
 </constraint>
</options>
```

Install the options using the `/config/query` service, as described in “Creating or Modifying Query Options” on page 169. Use a command similar to the following:

```
$ curl --anyauth --user user:password -X PUT \
    -d @./suggest-options.xml -i -H "Content-type: application/xml" \
    http://localhost:8003/v1/config/query/opt-suggest
```

The database and REST API instance are now configured to retrieve the example search suggestions.

4.13.2.3 Retrieve Unconstrained Search Suggestions

The following request returns all values of `<name/>` that match the wildcard string “doc*”:

```
$ curl --anyauth --user admin:adm1n -X GET \
    -H "Accept: application/xml" \
  'http://localhost:8003/v1/suggest?partial-q=doc&options=opt-suggest'
...
<search:suggestions xmlns:search=...>
  <search:suggestion>document-insert</search:suggestion>
  <search:suggestion>document-load</search:suggestion>
  <search:suggestion>document-query</search:suggestion>
</search:suggestions>
```

If you perform the same request with an Accept header of `application/json`, the response data looks like the following:

```
{
  "suggestions": [
    "document-insert",
    "document-load",
    "document-query"
```

```
  ]
}
```

4.13.2.4 Retrieve Constrained Search Suggestions

Recall that the query options include a value constraint on the `prefix` element:

```
<options xmlns="http://marklogic.com/appservices/search">
 <default-suggestion-source>
   <range type="xs:string" facet="true">
      <element ns="http://marklogic.com/example" name="name"/>
   </range>
 </default-suggestion-source>
 <constraint name="prefix">
   <value>
      <element ns="http://marklogic.com/example" name="prefix"/>
   </value>
 </constraint>
</options>
```

The following command uses this constraint in the additional query `prefix:xdmp` to limit results to suggestions in fragments with the `prefix` value `xdmp`. This eliminates the function `document-query` from the results because it has the prefix `cts`.

```
$ curl --anyauth --user admin:adm1n -X GET \
    -H "Accept: application/xml" \
  'http://localhost:8003/v1/suggest?partial-q=doc&options=opt-suggest&
q=prefix:xdmp'
...
<search:suggestions xmlns:search=...>
  <search:suggestion>document-insert</search:suggestion>
  <search:suggestion>document-load</search:suggestion>
</search:suggestions>
```

If you perform the same request with an Accept header of `application/json`, the response data looks like the following:

```
{
  "suggestions": [
    "document-insert",
    "document-load"
  ]
}
```

4.13.3 Example: Generating Search Suggestions Using POST

This example uses a POST request with the following combined query to limit results to suggestions in fragments with the `prefix` value `xdmp`. The example assumes your database contains the documents loaded in “Initialize the Database” on page 201.

```
<search xmlns="http://marklogic.com/appservices/search" >
  <query>
```

```
    <value-query>
      <element ns="http://marklogic.com/example" name="prefix"/>
      <text>xdmp</text>
    </value-query>
  </query>
  <options xmlns="http://marklogic.com/appservices/search">
    <default-suggestion-source>
      <range type="xs:string" facet="true">
        <element ns="" name="name"/>
      </range>
    </default-suggestion-source>
  </options>
</search>
```

Note that you do not need to pre-install persistent query options in this case because the options are included in the query.

The following command uses this query to retrieve suggestions for the input string "doc".

```
$ curl --anyauth --user admin:adm1n -X GET \
    -H "Accept: application/xml" \
  'http://localhost:8003/v1/suggest?partial-q=doc
...
<search:suggestions xmlns:search=...>
  <search:suggestion>document-insert</search:suggestion>
  <search:suggestion>document-load</search:suggestion>
</search:suggestions>
```

The following query is the equivalent combined query, expressed as JSON:

```
{ "search": {
    "qtext": [ "document" ],
    "query": {
      "value-query": {
        "element": {
          "ns": "http://marklogic.com/example",
          "name": "prefix"
        },
        "text": [ "xdmp" ]
      }
    },
    "options": {
      "default-suggestion-source": {
        "range": {
          "type": "xs:string",
          "facet": true,
          "element": {
            "ns": "",
            "name": "name"
          }
        }
      }
    }
```

```
} }
```

If you perform the same request with an Accept header of `application/json`, the response data looks like the following:

```
{
  "suggestions": [
    "document-insert",
    "document-load"
  ]
}
```

4.13.4 Where to Find More Information

For more details on using search suggestions, including performance recommendations and additional examples, see the following:

- `search:suggest` (XQuery function)
- Search Term Completion Using search:suggest in *Search Developer's Guide*.
- The XML query options "<default-suggestion-source>" on page 377 and "<suggestion-source>" on page 387, or the JSON equivalent, "suggestion-source" on page 348.
- "Specifying Dynamic Query Options with Combined Query" on page 161

5.0 Reading and Writing Multiple Documents

The MarkLogic REST API includes interfaces that enable you to read or write multiple documents in a single request. You can select documents for a bulk read using a list of URIs or a query.

- Terms and Definitions
- Writing Multiple Documents
- Reading Multiple Documents by URI
- Reading Multiple Documents Matching a Query
- Bulk Read Response Overview

5.1 Terms and Definitions

This chapter uses the following terms and definitions:

Term	Definition
content part	A part of a `multipart/mixed` request body or response that contains document content. Content must be XML, JSON, Text, or Binary.
metadata part	A part of a `multipart/mixed` request body or response that contains document metadata such as properties, collections, quality, and permissions. The content in a metadata part must XML or JSON and use the syntax described in "Working with Metadata" on page 124.
system default metadata	The default metadata values configured into MarkLogic Server. There are no defaults for collections or properties.
request default metadata	During bulk write, request-specific metadata that applies to documents without document-specific metadata. Request default metadata supercedes system default metadata. For details, see "Constructing a Metadata Part" on page 220.
document-specific metadata	During bulk write, a metadata part that applies to a specific document. Document-specific metadata supercedes request default metadata and system default metadata. For details, see "Constructing a Metadata Part" on page 220.

5.2 Writing Multiple Documents

To write multiple documents in a single request, send a POST request to the `/v1/documents` service with a URL of the following form and set the Content-Type header to `multipart/mixed`.

```
http://host:port/version/documents
```

The request must not include a `uri` or `extension` request parameter and the Content-Type header value must be `multipart/mixed`. Each part in the request body contains either content or metadata, which is indicated by the Content-Disposition part header.

MarkLogic Server creates or updates the documents and metadata in a bulk write request in a single transaction. If a single insertion or update fails, the entire batch of document operations fails.

When you use bulk write, pre-existing document properties are preserved, but other categories of metadata are completely replaced. If you want to preserve pre-existing metadata, use a single document write, such as a PUT request to `/documents`. For details, see “Manipulating Documents” on page 46.

> **Note:** When running the examples in this section, do not cut and paste the example `multipart/mixed` POST bodies. A `multipart/mixed` request body contains control characters that are not preserved when you copy the text. For details, see “Generating Example Payloads with XQuery” on page 236.

For details, see the following topics.

- Example: Loading Multiple Documents
- Request Body Overview
- Response Overview
- Constructing a Content Part
- Understanding Metadata Scoping
- Understanding When Metadata is Preserved or Replaced
- Constructing a Metadata Part
- Applying a Write Transformation
- Example: Controlling Metadata Through Defaults
- Example: Adding Documents to a Collection
- Generating Example Payloads with XQuery

5.2.1 Example: Loading Multiple Documents

This example provides a quick introduction to using the bulk write feature. It creates an XML document and a JSON document.

The XML document uses the system default metadata since the document part is not preceded by any document-specific or request default metadata. The JSON document uses the document-specific metadata included in the request.

The POST body contains three parts: An XML content part for a document with URI `doc1.xml`, a JSON metadata part for a document with URI `doc2.json`, and a content part for a JSON document with URI `doc2.json`.

The part Content-Type header indicates the MIME type of the part contents. The part Content-Disposition header indicates whether a part contains content or metadata. For details, see "Request Body Overview" on page 210.

The following graphic shows the breakdown of the parts.

Example Multipart POST Body for Bulk Write

XML Content

```
--BOUNDARY
Content-Type: application/xml
Content-Disposition: attachment; filename="doc1.xml"

<?xml version="1.0" encoding="UTF-8"?>
<root><a>Some data in an XML document</a></root>
```

Document-Specific Metadata

```
--BOUNDARY
Content-Type: application/json
Content-Disposition: attachment; filename="doc2.json";
category=metadata

{"quality" : 2 }
```

JSON Content

```
--BOUNDARY
Content-Type: application/json
Content-Disposition: attachment; filename="doc2.json"

{"key":"value"}
--BOUNDARY--
```

You cannot create a working multipart POST body by cutting and pasting from the examples in this guide. If your development environment does not include other tools or libraries for constructing a multipart HTTP request body, you can use the following XQuery to generate the payload using the technique described in "Generating Example Payloads with XQuery" on page 236.

```
xquery version "1.0-ml";

declare variable $OUTPUT-FILENAME := "/example/simple-body";
declare variable $BOUNDARY := "BOUNDARY";
```

```
let $xml-doc:= document{ <root><a>Some data in an XML document</a></root> }
let $json-metadata := text { '{"quality" : 2 }'}
let $json-doc :=text{'{"key":"value"}'}
let $manifest :=
   <manifest>
     <!-- doc1.xml content -->
     <part>
       <headers>
          <Content-Type>application/xml</Content-Type>
          <Content-Disposition>attachment;
filename="doc1.xml"</Content-Disposition>
       </headers>
     </part>
     <!-- doc2.json metadata -->
     <part>
       <headers>
          <Content-Type>application/json</Content-Type>
          <Content-Disposition>attachment; filename="doc2.json";
category=metadata</Content-Disposition>
       </headers>
     </part>
     <!-- doc2.json content -->
     <part>
       <headers>
         <Content-Type>application/json</Content-Type>
         <Content-Disposition>attachment;
filename="doc2.json"</Content-Disposition>
       </headers>
     </part>
   </manifest>
return
xdmp:save("$OUTPUT-FILENAME",
  xdmp:multipart-encode(
    $BOUNDARY, $manifest, ($xml-doc,$json-metadata,$json-doc)
  )
)
```

The following command writes the documents and receives a JSON response. Use the Accept header to control whether the request returns an XML or JSON response. The boundary in the header must match the boundary that separates the parts in the request body.

```
$ curl --anyauth --user user:password -X POST \
  --data-binary @/example/simple-body -i \
  -H "Accept: application/json" \
  -H "Content-type: multipart/mixed; boundary=BOUNDARY" \
  http://localhost:8003/v1/documents
```

MarkLogic Server responds with the following data. The response includes a `document` entry for each created or updated document. Each `document` entry provides enough information to retrieve the associated document and/or its metadata. For details, see “Response Overview” on page 211.

```
HTTP/1.1 200 OK
Server: MarkLogic
Content-Type: text/xml; charset=UTF-8
Content-Length: 449
```

```
Connection: Keep-Alive
Keep-Alive: timeout=5

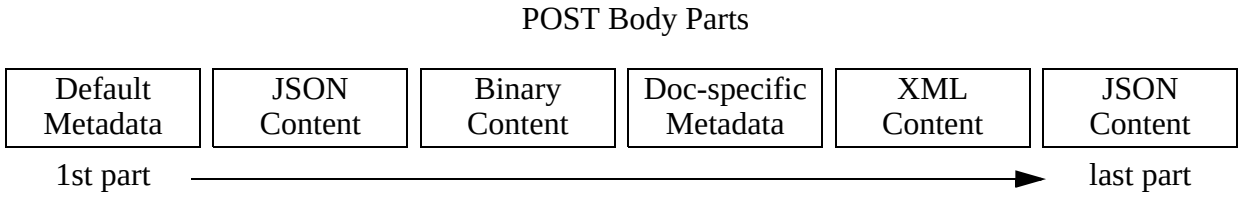
<rapi:documents xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:document>
    <rapi:uri>doc1.xml</rapi:uri>
    <rapi:category>metadata</rapi:category>
    <rapi:category>content</rapi:category>
    <rapi:mime-type>application/xml</rapi:mime-type>
  </rapi:document>
  <rapi:document>
    <rapi:uri>doc2.json</rapi:uri>
    <rapi:category>metadata</rapi:category>
    <rapi:category>content</rapi:category>
    <rapi:mime-type>application/json</rapi:mime-type>
  </rapi:document>
</rapi:documents>
```

5.2.2 Request Body Overview

The multipart body of a bulk write request represents a "stream" of document metadata and/or content. Each part contains either content or metadata. The request body has the following key features:

- Document content can be heterogeneous. You can create any combination of XML, JSON, Text, and Binary documents in a single request.
- Metadata can be expressed in XML or JSON.
- You can create documents with the system default metadata, request default metadata, or document-specific metadata. You can mix these metadata sources in the same request.

For example, a request with the following parts in the POST body creates 2 JSON documents, a binary document, and an XML document. The XML document has document-specific metadata, while the other documents use the request default metadata from the first part.



For a complete example, see "Example: Loading Multiple Documents" on page 208.

The part headers describe the kind of part (content or metadata), the MIME type of the part body, and additional part-specific options. The table below outlines the role of the headers for content and metadata parts.

Part Type	Header	Description
Content	Content-Type	Determines the document type (JSON, XML, Text, or Binary).
	Content-Disposition	Identifies the part as a content part. Contains additional parameters that specify the document URI and options such as XML repair.
Metadata	Content-Type	Specifies the metadata MIME type. Metadata must be JSON or XML.
	Content-Disposition	Identifies the part as a metadata part. Determines whether the metadata is document-specific metadata or request default metadata.

For details, see the following topics:

- “Constructing a Content Part” on page 213
- “Constructing a Metadata Part” on page 220

5.2.3 Response Overview

If a bulk write request is successful, the response body contains a list of document descriptors, one per document created, in the order of creation. The response can be XML or JSON, depending on the Accept header.

If there is an error creating one of the documents, the entire batch is rejected and the database is unchanged. In the event of an error, MarkLogic Server responds with error status code and the response body includes details about the failing document.

For example, the following response data reflects successful creation of 3 documents. Both metadata and content are available for `doc1.xml` and `doc2.json`, but only metadata is availble for `doc3.xml`.

Format	Response Data
XML	<rapi:documents xmlns:rapi="http://marklogic.com/rest-api"> <rapi:document> <rapi:uri>**doc1.xml**</rapi:uri> <rapi:category>metadata</rapi:category> <rapi:category>content</rapi:category> <rapi:mime-type>application/xml</rapi:mime-type> </rapi:document> <rapi:document> <rapi:uri>**doc2.json**</rapi:uri> <rapi:category>metadata</rapi:category> <rapi:category>content</rapi:category> <rapi:mime-type>application/json</rapi:mime-type> </rapi:document> <rapi:document> <rapi:uri>**doc3.xml**</rapi:uri> <rapi:category>metadata</rapi:category> <rapi:mime-type>application/xml</rapi:mime-type> </rapi:document> </rapi:documents>

JSON	{ "documents": [
	  { "uri": "doc1.xml",
	  "mime-type": "application/xml",
	  "category": [
	  "metadata",
	  "content"
	  ]
	  },
	  { "uri": "doc2.json",
	  "mime-type": "application/json",
	  "category": [
	  "metadata",
	  "content"
	  ]
	  },
	  { "uri": "doc3.xml",
	  "mime-type": "application/xml",
	  "category": [
	  "metadata",
	  "content"
	  ]
	  }
	] }

You can use the data in the response to retrieve the inserted documents. For example, you can retrieve `doc2.json` using the following GET request, where the MIME type in the Accept header comes from the response `mime-type` element/key value and the `uri` and `category` request parameter values come from the corresponding response element/key values.

```
$ curl --anyauth --user user:password -X GET -i \
  -H "Accept: application/json" \
  http://localhost:8003/v1/documents?uri=doc2.json&category=content
```

The following table provides more details on the information returned about each document:

Element or Key Name	Description
`uri`	The database URI of the document. This is either the URI supplied in the POST body or a server-generated URI. You can use this data to build the `uri` parameter of a document retrieval request such as `GET /v1/documents`.
`category`	The categories of data updated (`content` or `metadata`). You can use this data to build the `category` parameter of a document retrieval request, such as `GET /v1/documents`. If the request includes a content update for a document, both `content` and `metadata` are returned. If only metadata is updated, then only `metadata` is returned.
`mime-type`	A MIME type suitable for retrieving the contents of the document. For example, you can use this data in the Accept header of a document retrieval request such as `GET /v1/documents`.

5.2.4 Constructing a Content Part

When constructing a content part, you must include a Content-Type header that controls the document type and a Content-Disposition header that controls the document URI. The Content-Disposition header can also include additional options.

A content part is distinguished from a metadata part by the absence of `category=metadata` in the Content-Disposition header.

For details, see the following topics:

- Controlling Document Type
- Specifying an Explicit Document URI

- Automatically Generating a Document URI
- Adding Content Options
- Example Content Part Headers

5.2.4.1 Controlling Document Type

You can create JSON, XML, Binary, and Text documents in a bulk write. The MIME type of the part Content-Type header determines the database document type.

> **Note:** You are responsible for ensuring the MIME type and the part contents match. MarkLogic Server does not sniff the contents or perform implicit content conversions.

The MarkLogic Server MIME type mapping determines the document type for XML, Text, and Binary documents. For example, the following header creates an XML document because the MIME type `application/xml` is mapped to XML in the default server MIME type mapping.

```
Content-Type: application/xml
```

To review or change the MIME type mapping, see the "Mimetypes" subsection of the Groups section of the Admin Interface.

For JSON documents, content is assumed to be JSON if the MIME type begins with `application/`, optionally followed by `some-format+`, and ends with `json`. For example, all the following headers indicate a part that contains JSON:

```
Content-Type: application/json
Content-Type: application/rdf+json
```

5.2.4.2 Specifying an Explicit Document URI

To include an explicit URI for a document, set the content disposition of the metadata and/or content part to `attachment` and put the URI in the `filename` parameter. That is, use the following header template:

```
Content-Disposition: attachment;filename=/your/uri
```

A metadata part of the above form signifies document-specific metadata. If the request includes content for the same document, the corresponding content part must immediately follow the document-specific metadata part. For details, see "Constructing a Metadata Part" on page 220.

For examples, see "Example Content Part Headers" on page 215.

5.2.4.3 Automatically Generating a Document URI

To have MarkLogic Server generate the URI for a document, set the content disposition to `inline` and include an `extension` parameter that specifies the URI extension. You can also include an optional `directory` parameter that specifies a destination database directory. That is, use one of the following header templates:

```
Content-Disposition: inline;extension=suffix
Content-Disposition: inline;extension=suffix;directory=/your/dir/
```

Note: Do not include a separator in the extension value. MarkLogic Server will prefix your suffix with a period (.).

Note: A directory path must end with a slash (/).

Best practice is to use an extension suffix for which there is a MIME type mapping in MarkLogic Server. The part Content-type should match the MIME type corresponding to the extension. To review or change the MIME type mapping, see the "Mimetypes" subsection of the Groups section of the Admin Interface.

You can only use server-generated URIs on content parts. Metadata for documents with server-assigned URIs can only come from request default metadata or system default metadata. For details, see "Constructing a Metadata Part" on page 220.

For example, the following headers signify a JSON content part with a server-generated URI that uses the directory `/my/dir` and ends with `.json`, such as `/my/dir/1234567890.json`.

```
Content-type: application/json
Content-Disposition: inline;extension=json;directory=/my/dir/
```

For more examples, see "Example Content Part Headers" on page 215.

5.2.4.4 Adding Content Options

Use the Content-Disposition header to include document-specific options such as `lang`, `repair`, or `extract`. For example, the use of `extract=properties` in the following header tells MarkLogic Server to extract metadata from a binary document and save it as a document property.

```
Content-type: image/jpeg
Content-Disposition: attachment;filename=my.jpg;extract=properties
```

For option details, refer to the API reference for `POST:/v1/documents`.

5.2.4.5 Example Content Part Headers

The table below shows example headers for several kinds of content part.

Content Part Description	Example Headers
XML document with an explicit URI	`Content-type: application/xml` `Content-Disposition: attachment;filename=/my/uri.xml`
XML document with a repair option	`Content-type: application/xml` `Content-Disposition: attachment;filename=my.xml;repair=none`
JSON document with an explicit URI	`Content-type: application/json` `Content-Disposition: attachment;filename=/my/uri.json`
JSON document with a server assigned URI and a client-specified database directory	`Content-type: application/json` `Content-Disposition: inline;extension=json;directory=/my/dir/`
Binary document with a server assigned URI and a client specified database directory	`Content-type: image/jpeg` `Content-Disposition: inline;extension=jpg;directory=/images/`

5.2.5 Understanding Metadata Scoping

This topic describes how metadata is selected for documents created or updated with a bulk write. For details on the structure of a metadata part, see “Constructing a Metadata Part” on page 220.

Note: For performance reasons, pre-existing metadata other than properties is completely replaced during a bulk write operation, either with values supplied in the request or with system defaults.

Metadata in a bulk write can be drawn from 3 possible sources, as shown in the table below. The table lists the metadata sources from highest to lowest precedence, so a source supercedes those below it if both are present.

Metadata Type	Description
document-specific metadata	A metadata part that includes a document URI in its headers applies to a single document. If the request includes content for the document, the content part must occur immediately after the metadata part.
request default metadata	A metadata part that does not include a document URI in its headers is default metadata that applies to all subsequent content parts that do not have docment-specific metadata. The default applies until the occurrence of the next request default metadata part or the end of the request body.
system default metadata	Default metadata configured into MarkLogic server. This metadata applies any time no document-specific or request default metadata are in scope.

The following rules determine what metadata applies during document creation.

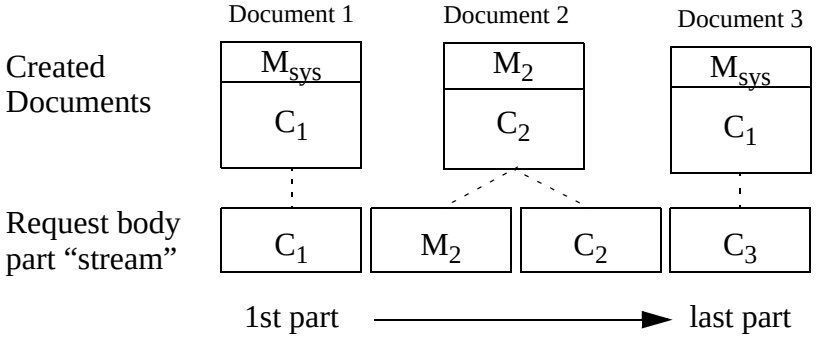
- If a content part is immediately preceded by a corresponding document-specific metadata part, use it.
- If a content part is not immediately preceded by document-specific metadata, then use the most recent preceding request default metadata part.
- If there is no request default metadata and no document-specific metadata, then use the system default metadata.
- A request metadata part entirely replaces previous request metadata. For performance reasons, no merging occurs.
- A metadata category not included in the part body is either set to the system default metadata value or left unchanged, depending upon whether or not the request includes a content update. For details, see “Understanding When Metadata is Preserved or Replaced” on page 219.

  Note: If a request body includes both content and document-specific metadata for the same document, the metadata part must immediately precede the content part.

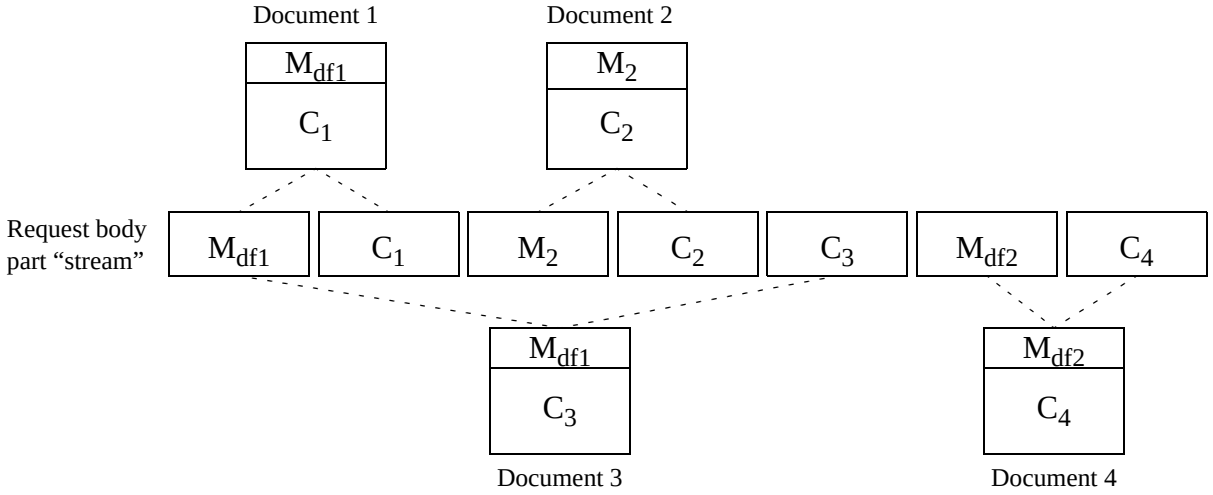
Each occurrence of request default metadata completely replaces any preceding request default metadata. Similarly, request default metadata and document-specific metadata are not merged. For example, if a document-specific metadata part contains only a collections setting, it inherits quality, permissions and properties from the system default metadata, not from any preceding request default metadata.

The following examples illustrate application of these rules. In these examples, C_n represents a content part for the Nth document, M_n represents document-specific metadata for the Nth document, M_{dfn} represents the Nth occurrence of request default metadata, and M_{sys} is the system default metadata.

The following input creates 3 documents. Documents 1 and Document 3 use system default metadata. Document 2 uses document-specific metadata.



The following input creates four documents, using a combination of request default metadata and document-specific metadata. Document 1, Document 3, and Document 4 use request default metadata. Document 2 uses document-specific metadata. Document 1 and Document 3 use the first block of request default metadata, M_{df1}. After Document 3 is created, M_{df2} replaces M_{df1} as the request default metadata, so Document 4 uses the metadata in M_{df2}.



5.2.6 Understanding When Metadata is Preserved or Replaced

This topic discusses when a bulk write preserves or replaces pre-existing metadata. You can skip this section if your bulk write operations only create new documents or you do not need to preserve pre-existing metadata such as permissions, document quality, collections, and properties.

When there is no request default metadata and no document-specific metadata, all metadata categories other than properties are set to the system default values. Properties are unchanged.

In all other cases, either request default metadata or document-specific metadata is used when creating a document, as described in “Understanding Metadata Scoping” on page 216.

When you update both content and metadata for a document in the same bulk write request, the following rules apply, whether applying request default metadata or document-specific metadata:

- The metadata in scope is determined as described in “Understanding Metadata Scoping” on page 216.
- Any metadata category that has a value in the in-scope metadata completely replaces that category.
- Any metadata category other than properties that is missing or empty in the in-scope metadata is completely replaced by the system default value.
- If the in-scope metadata does not include properties, then existing properties are preserved.
- If the in-scope metadata does not include collections, then collections are reset to the default. There is no system default for collections, so this results in a document being removed from all collections if no default collections are specified for the user role performining the update.

When you include a document-specific metadata part, but no content part, you update only the metadata for a document. In this case, the following rules apply:

- Any metadata category that has a value in the document-specific metadata completely replaces that category.
- Any metadata category that is missing or empty in the document-specific metadata is preserved.

The table below shows how pre-existing metadata changes if a bulk write updates just the content, just the collections metadata (via document-specific metadata), or both content and collections metadata (via request default metadata or document-specific metadata).

Metadata Category	Update Content Only	Update Metadata Only	Update Content & Metadata
collections	reset	modified to new value	modified to new value
quality	reset	preserved	reset
permissions	reset	preserved	reset
properties	preserved	preserved	preserved

The results are similar if the metadata update modifies other metadata categories.

5.2.7 Constructing a Metadata Part

This section describes how metadata values are applied during a bulk write and how to construct a request default or document-specific metadata part. Metadata must expressed as XML or JSON and must be specified using the syntax described in “Working with Metadata” on page 124.

The following topics are covered:

- Constructing a Request Default Metadata Part
- Constructing a Document-Specific Metadata Part

5.2.7.1 Constructing a Request Default Metadata Part

A request default metadata part applies to all documents created from the occurrence of the metadata part until the next request default metadata part or the end of the POST body, except for documents that have document-specific metadata.

A request default metadata part must use headers of the following form, where *metadata-MIME-type* is a MIME type that maps to XML or JSON metadata content:

```
Content-Type: metadata-MIME-type
Content-Disposition: inline; category=metadata
```

For example, use following headers on a default metadata part that contains JSON metadata:

```
Content-Type: application/json
Content-Disposition: inline; category=metadata
```

The part contents must be document metadata of the form described in “Working with Metadata” on page 124.

For a complete example, see “Example: Controlling Metadata Through Defaults” on page 225.

5.2.7.2 Constructing a Document-Specific Metadata Part

Document-specific metadata applies to a single document. The part must use headers of the following form, where *`metadata-MIME-type`* is a MIME type that maps to XML or JSON metadata content and *`document-uri`* is the database URI of the target document.

```
Content-Type: metadata-MIME-type
Content-Disposition: attachment; filename=document-uri; category=metadata
```

If both content and metadata for the document are included in the POST request, then the document-specific metadata part must occur immediately before the content part.

If the input includes document specific metadata without an accompanying content part, the document must already exist. Only the metadata of the document changes.

Any metadata sub-category other than properties that is not included in the part body is set to the request default value, if one is in scope. Otherwise, it is set to the system default value. However, there is no system default for collections or properties. Existing properties are unchanged if not included. For collections, if the user performing the update has a role that includes default collections, then that default is used. If there are no default collections for the role, then the document is removed from all collections.

> **Note:** When updating permissions, you need not include the default `rest-reader` and `rest-writer` roles.

For example, suppose a pre-exisitng document has the following customized metadata:

- document quality of 2
- membership in a collection named “April 2014”
- read permissions for a custom role
- a custom property named “custom-prop”.

That is, if you retrieve the metadata for this document using `GET /version/documents`, then the response contains metadata similar to the following. The metadata values that differ from the system default are shown in bold.

Format	Metadata

XML	<rapi:metadata xmlns:rapi="http://marklogic.com/rest-api"...>
	  <rapi:collections>
	  <rapi:collection>April 2014</rapi:collection>
	  </rapi:collections>
	  <rapi:permissions>
	  <rapi:permission>
	  <rapi:role-name>rest-writer</rapi:role-name>
	  <rapi:capability>update</rapi:capability>
	  </rapi:permission>
	  <rapi:permission>
	  <rapi:role-name>rest-reader</rapi:role-name>
	  <rapi:capability>read</rapi:capability>
	  </rapi:permission>
	  **<rapi:permission>
	  <rapi:role-name>some-custom-role</rapi:role-name>
	  <rapi:capability>read</rapi:capability>
	  </rapi:permission>**
	  </rapi:permissions>
	  <prop:properties xmlns:prop="http://marklogic.com/xdmp/property">
	  <custom-prop>some property value</custom-prop>
	  <prop:last-modified>2014-04-10T16:11:56-07:00</prop:last-modified>
	  </prop:properties>
	  <rapi:quality>2</rapi:quality>
	</rapi:metadata>

Format	Metadata

JSON	<pre>{
	  "collections": ["April 2014"],
	  "permissions": [
	  {
	  "role-name": "rest-writer",
	  "capabilities": ["update"]
	  },
	  {
	  "role-name": "rest-reader",
	  "capabilities": ["read"]
	  },
	  {
	  "role-name": "custom-role",
	  "capabilities": ["read"]
	  }
	  ],
	  "properties": {
	  "custom-prop": "some property value",
	  "$ml.prop": { "last-modified": "2014-04-10T16:21:15-07:00" }
	  },
	  "quality": 2
	}</pre>

If you use a bulk write to update the metadata for this document, specifying only a new document quality of 1, then the permissions and collections are reset to the system default, but the custom property is preserved. Assuming out-of-the box MarkLogic Server system defaults, the updated document is no longer be in any collections, and the read permission for “custom-role” is removed, resulting in the following metadata:

Format	Metadata

XML	<rapi:metadata xmlns:rapi="http://marklogic.com/rest-api"...>
	  <rapi:collections/>
	  <rapi:permissions>
	  <rapi:permission>
	  <rapi:role-name>rest-writer</rapi:role-name>
	  <rapi:capability>update</rapi:capability>
	  </rapi:permission>
	  <rapi:permission>
	  <rapi:role-name>rest-reader</rapi:role-name>
	  <rapi:capability>read</rapi:capability>
	  </rapi:permission>
	  </rapi:permissions>
	  <prop:properties xmlns:prop="http://marklogic.com/xdmp/property">
	  <custom-prop>some property value</custom-prop>
	  <prop:last-modified>2014-04-10T16:11:56-07:00</prop:last-modified>
	  </prop:properties>
	  <rapi:quality>1</rapi:quality>
	</rapi:metadata>

JSON	{
	  "collections": [],
	  "permissions": [
	  {
	  "role-name": "rest-writer",
	  "capabilities": ["update"]
	  },
	  {
	  "role-name": "rest-reader",
	  "capabilities": ["read"]
	  }
	  ],
	  "properties": {
	  "custom-prop": "some property value",
	  "$ml.prop": { "last-modified": "2014-04-10T16:21:15-07:00" }
	  },
	  "quality": 1
	}

5.2.8 Applying a Write Transformation

A content write transformation is a user-defined XQuery function or XSLT style sheet used to modify documents during insertion, as described in “Working With Content Transformations” on page 269.

You can apply a content transformation to all documents created by a bulk write request by making a POST request with a URL of the following form:

```
http://host:port/version/documents?transform=name&trans:param=value
```

For example, the following URL specifies a transform function named `example` with a value of “me” for the function parameter named `reviewer`.

```
http://myhost:8003/v1/documents?transform=example&trans:reviewer=me
```

Transform parameters are optional, depending on the transform interface. The same transform parameter values are applied to every document document in the request.

MarkLogic Server applies the transformation prior to inserting each document into the database. Your transform can choose not to make any modifications, but it will be invoked for every document.

You cannot specify more than one transform per request. However, your transform function can import and call other transformation functions to get the same effect.

For details, see “Working With Content Transformations” on page 269.

5.2.9 Example: Controlling Metadata Through Defaults

This example creates documents that use the system default, request default, and document specific metadata, as described in “Constructing a Metadata Part” on page 220.

- Payload Description
- Generating the POST Body
- Executing the Request
- Verifying the Results

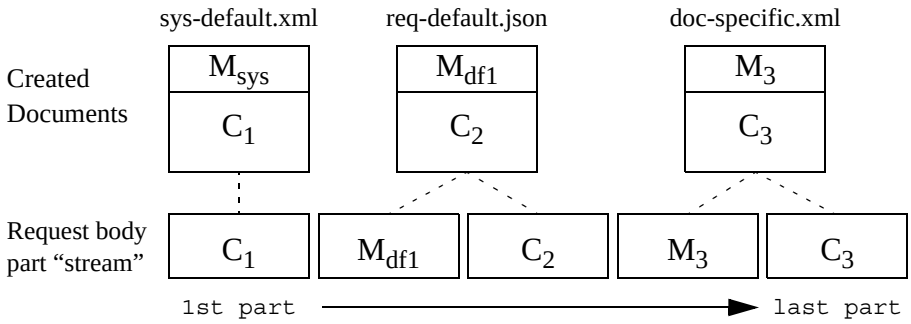
5.2.9.1 Payload Description

This example uses document quality to illustrate how default metadata affects the documents you create. The document quality setting used in this example result in creation of the following documents:

- `sys-default.xml` with document quality 0, from the system default metadata
- `req-default.json` with document quality 2, from M_{df1}

- `doc-specific.xml` with document quality 1, from M_3

The following picture represents the parts in the request body and the documents created from them. In the picture, M_n represents metadata, C_n represents content. Note that the metadata is not literally embedded in the created documents; content and metadata are merely grouped here for illustrative purposes.



The following multipart body implements the pictured bulk write input. The annotations on the right indicate the start of each part in the graphic above. These annotations are not present in the actual POST body.

```
--BOUNDARY
Content-Type: application/xml                                    (C1)
Content-Disposition: attachment; filename="sys-default.xml"
Content-Length: 86

<?xml version="1.0" encoding="UTF-8"?>
<root>a doc with system default metadata</root>
--BOUNDARY
Content-Type: application/json                                   (Mdf1)
Content-Disposition: inline; category=metadata
Content-Length: 16

{"quality" : 2 }
--BOUNDARY
Content-Type: application/json                                   (C2)
Content-Disposition: attachment; filename="req-default.json"
Content-Length: 45

{"key":"a doc with request default metadata"}
--BOUNDARY
Content-Type: application/xml                                    (M3)
Content-Disposition: attachment; filename="doc-specific.xml";
category=metadata
Content-Length: 147

<?xml version="1.0" encoding="UTF-8"?>
<rapi:metadata xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:quality>1</rapi:quality>
```

```
</rapi:metadata>
--BOUNDARY
Content-Type: application/xml                                          (C3)
Content-Disposition: attachment; filename="doc-specific.xml"
Content-Length: 89

<?xml version="1.0" encoding="UTF-8"?>
<root>a doc with document-specific metadata</root>
--BOUNDARY--
```

5.2.9.2 Generating the POST Body

This section contains an XQuery query you can use to generate the POST body, using the technique described in as described in "Generating Example Payloads with XQuery" on page 236. Skip this section if you use other tools to generate the request body.

Before running this query in Query Console, modify `$OUTPUT-FILENAME` for your environment:

```
xquery version "1.0-ml";

declare variable $OUTPUT-FILENAME := "/example/default_metadata";
declare variable $BOUNDARY := "BOUNDARY";

let $sys-default:=
  document{ <root>a doc with system default metadata</root> }
let $req-specific :=
  text{'{"key":"a doc with request default metadata"}' }
let $doc-specific :=
  document{ <root>a doc with document-specific metadata</root> }
let $req-metadata := text { '{"quality" : 2 }'}
let $sys-default-json :=
  text { '{ "key": "a doc with system default metadata" }' }
let $empty-metadata := text { '{ }' }
let $doc-metadata := document {
    <rapi:metadata xmlns:rapi="http://marklogic.com/rest-api">
      <rapi:quality>1</rapi:quality>
    </rapi:metadata>
  }
let $manifest :=
   <manifest>
     <!-- content, using system default metadata -->
     <part>
       <headers>
          <Content-Type>application/xml</Content-Type>
          <Content-Disposition>attachment;
filename="sys-default.xml"</Content-Disposition>
       </headers>
     </part>
     <!-- Request default metadata -->
     <part>
       <headers>
          <Content-Type>application/json</Content-Type>
          <Content-Disposition>inline;
```

```
category=metadata</Content-Disposition>
       </headers>
     </part>
     <!-- JSON document using request default metadata -->
     <part>
       <headers>
          <Content-Type>application/json</Content-Type>
          <Content-Disposition>attachment;
filename="req-default.json"</Content-Disposition>
       </headers>
     </part>
      <!-- document specific metadata, in XML -->
      <part>
       <headers>
          <Content-Type>application/xml</Content-Type>
         <Content-Disposition>attachment; filename="doc-specific.xml";
category=metadata</Content-Disposition>
       </headers>
     </part>
     <!-- XML document with document specific metadata -->
     <part>
       <headers>
          <Content-Type>application/xml</Content-Type>
          <Content-Disposition>attachment;
filename="doc-specific.xml"</Content-Disposition>
       </headers>
     </part>
   </manifest>
return
xdmp:save($OUTPUT-FILENAME,
  xdmp:multipart-encode(
    $BOUNDARY, $manifest,
    ($sys-default,
     $req-metadata,
     $req-specific,
     $doc-metadata,
     $doc-specific)
  )
)
```

5.2.9.3 Executing the Request

The following command executes the bulk write request, assuming `/example/default_metadata` contains the the POST body. Modify the value of the `--data-binary` option to the file that contains your POST body.

```
$ curl --anyauth --user user:password -X POST -i \
  --data-binary @/example/default_metadata \
  -H "Content-type: multipart/mixed; boundary=BOUNDARY" \
  http://localhost:8003/v1/documents
```

MarkLogic Server returns output similar to the following. For details, see “Response Overview” on page 211.

```
HTTP/1.1 200 OK
Server: MarkLogic
Content-Type: text/xml; charset=UTF-8
Content-Length: 656
Connection: Keep-Alive
Keep-Alive: timeout=5

<?xml version="1.0"?>
<rapi:documents xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:document>
    <rapi:uri>sys-default.xml</rapi:uri>
    <rapi:category>metadata</rapi:category>
    <rapi:category>content</rapi:category>
    <rapi:mime-type>application/xml</rapi:mime-type>
  </rapi:document>
  <rapi:document>
    <rapi:uri>req-default.json</rapi:uri>
    <rapi:category>metadata</rapi:category>
    <rapi:category>content</rapi:category>
    <rapi:mime-type>application/json</rapi:mime-type>
  </rapi:document>
  <rapi:document>
    <rapi:uri>doc-specific.xml</rapi:uri>
    <rapi:category>metadata</rapi:category>
    <rapi:category>content</rapi:category>
    <rapi:mime-type>application/xml</rapi:mime-type>
  </rapi:document>
</rapi:documents>
```

For a JSON response, set the Accept header to `application/json`:

```
$ curl --anyauth --user user:password -X POST -i \
  --data-binary @/example/default_metadata \
  -H "Content-type: multipart/mixed; boundary=BOUNDARY" \
  -H "Accept: application/json" \
  http://localhost:8003/v1/documents
```

5.2.9.4 Verifying the Results

You can retrieve the document quality for the created documents individually, or you can use the following bulk read command to retrieve the quality for all the documents in a single request. This example requests JSON metadata; you can also retrieve metadata as XML. MarkLogic Server returns a `multipart/mixed` response where each response body part contains the requested data for a different document. For details, see “Reading Multiple Documents by URI” on page 239.

```
$ curl --anyauth --user user:password -X GET -i \
  -H "Accept: multipart/mixed; boundary=BOUNDARY" \
  'http://localhost:8004/v1/documents?category=quality&format=json&uri
=sys-default.xml&uri=req-default.json&uri=doc-specific.xml'
...
HTTP/1.1 200 OK
Content-type: multipart/mixed; boundary=BOUNDARY
```

```
Server: MarkLogic
Content-Length: 701
Connection: Keep-Alive
Keep-Alive: timeout=5

--BOUNDARY
Content-Type: application/json
Content-Disposition: attachment; filename=sys-default.xml;
category=quality; format=json
Content-Length: 13

{"quality":0}
--BOUNDARY
Content-Type: application/json
Content-Disposition: attachment; filename=req-default.json;
category=quality; format=json
Content-Length: 13

{"quality":2}
--BOUNDARY
Content-Type: application/json
Content-Disposition: attachment; filename=doc-specific.xml;
category=quality; format=json
Content-Length: 13

{"quality":1}
--BOUNDARY--
```

5.2.10 Example: Adding Documents to a Collection

This example demonstrates using request default metadata to add all documents to the same collection during insertion.

You can add selected documents to a different collection using document-specific metadata or by including an additional request default metadata part that uses a different collection; see “Example: Controlling Metadata Through Defaults” on page 225.

Since the metadata in the example request only includes settings for collections metadata, other metadata categories such as permissions and quality use the system default settings. For an example of more complex metadata, see “Extending the Example” on page 236.

For more information on metadata syntax, see “Working with Metadata” on page 124.

- Payload Description
- Generating the POST Body
- Executing the Request
- Verifying the Results
- Extending the Example

5.2.10.1 Payload Description

The payload contains 3 parts: A metadata part that specifies a collection and JSON and XML content parts for documents that are placed into that collection. The documents are added to the collection “April 2014”. The metadata is expressed as JSON. Both documents use server-assigned URIs.

	Multipart/mixed Request Body
Request Default Metadata	--BOUNDARY Content-Type: application/json Content-Disposition: inline; category=metadata Content-Length: 33 {"collections" : ["April 2014"] }
JSON Content	--BOUNDARY Content-Type: application/json Content-Disposition: inline; extension=json; directory=/example/ Content-Length: 18 { "key": "value" }
XML Content	--BOUNDARY Content-Type: application/xml Content-Disposition: inline; extension=xml; directory=/example/ Content-Length: 64 <?xml version="1.0" encoding="UTF-8"?> <root>some content</root> --BOUNDARY--

To use XML metadata instead of JSON, modify the Content-Type of the first part to `application/xml` and change the part contents to the following:

```
<rapi:metadata xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:collections>
    <rapi:collection>April 2014</rapi:collection>
  </rapi:collections>
</rapi:metadata>
```

5.2.10.2 Generating the POST Body

This section contains an XQuery query you can use to generate the POST body, using the technique described in as described in “Generating Example Payloads with XQuery” on page 236. Skip this section if you choose to use other tools to generate the request body.

Before running this query in Query Console, modify `$OUTPUT-FILENAME` for your environment:

```
xquery version "1.0-ml";
```

```
declare variable $OUTPUT-FILENAME := "/example/coll-body";
declare variable $BOUNDARY := "BOUNDARY";

let $xml-content := document{ <root>some content</root> }
let $metadata := text { '{"collections" : ["April 2014"] }'}
let $json-content := text { '{ "key": "value" }' }
let $manifest :=
   <manifest>
     <!-- Request default metadata -->
     <part>
       <headers>
          <Content-Type>application/json</Content-Type>
          <Content-Disposition>inline;
category=metadata</Content-Disposition>
       </headers>
     </part>
     <!-- JSON document -->
     <part>
       <headers>
          <Content-Type>application/json</Content-Type>
          <Content-Disposition>inline; extension=json;
directory=/example/</Content-Disposition>
       </headers>
     </part>
     <!-- XML document -->
     <part>
       <headers>
          <Content-Type>application/xml</Content-Type>
          <Content-Disposition>inline; extension=xml;
directory=/example/</Content-Disposition>
       </headers>
     </part>
     </manifest>
return
xdmp:save($OUTPUT-FILENAME,
  xdmp:multipart-encode(
    $BOUNDARY, $manifest, ($metadata,$json-content,$xml-content)
  )
)
```

To use XML metadata instead of JSON metadata, replace the value of `$metadata` with the following declaration and change the Content-Type header for the first part to `application/xml`.

```
<rapi:metadata xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:collections>
    <rapi:collection>April 2014</rapi:collection>
  </rapi:collections>
</rapi:metadata>
```

5.2.10.3 Executing the Request

Use the following command to create the two documents. MarkLogic Server responds with status code 200 OK and includes the documents URIs and additional information about the created documents in the response body, as XML.

```
$ curl --anyauth --user user:password -X POST -i \
  --data-binary @/example/coll-body \
  -H "Content-type: multipart/mixed; boundary=BOUNDARY" \
  http://localhost:8003/v1/documents
...
HTTP/1.1 200 OK
Server: MarkLogic
Content-Type: text/xml; charset=UTF-8
Content-Length: 498
Connection: Keep-Alive
Keep-Alive: timeout=5
<?xml version="1.0"?>
<rapi:documents xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:document>
    <rapi:uri>/example/11572018736249878991.json</rapi:uri>
    <rapi:category>metadata</rapi:category>
    <rapi:category>content</rapi:category>
    <rapi:mime-type>application/json</rapi:mime-type>
  </rapi:document>
  <rapi:document>
    <rapi:uri>/example/5255172137182404803.xml</rapi:uri>
    <rapi:category>metadata</rapi:category>
    <rapi:category>content</rapi:category>
    <rapi:mime-type>application/xml</rapi:mime-type>
  </rapi:document>
</rapi:documents>
```

To get JSON results, set the Accept header to `application/json` or use the `format=json` request parameter. The following is an example of the equivalent JSON response data:

```
{ "documents": [
    { "uri": "/example/11572018736249878991.json",
      "mime-type": "application/json",
      "category": [
        "metadata",
        "content"
      ]
    },
    { "uri": "/example/5255172137182404803.xml",
      "mime-type": "application/xml",
      "category": [
        "metadata",
        "content"
      ]
    }
] }
```

For details, see “Response Overview” on page 211.

5.2.10.4 Verifying the Results

You can use the following combined query with the `/search` service to verify the documents added to the “April 2014” collection, or use the Explore feature of Query Console to examine the database contents. For more information, see “Specifying Dynamic Query Options with Combined Query” on page 161 or the *Query Console User's Guide*.

```
$ cat coll-query.xml
<search xmlns="http://marklogic.com/appservices/search">
  <query>
    <collection-query>
      <uri>April 2014</uri>
    </collection-query>
  </query>
  <options>
    <transform-results apply="empty-snippet" />
    <return-metrics>false</return-metrics>
  </options>
</search>

$ curl --anyauth --user user:password -X POST -i \
  -H "Content-type: application/xml"
  -H "Accept: application/xml" -d @./coll-query.xml \
  http://localhost:8003/v1/search
```

You should get output similar to the following. The search results contain a match for the two documents previously created.

Format	Search Results
XML	<search:response snippet-format="empty-snippet" total="2" start="1" page-length="10" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="" xmlns:search="http://marklogic.com/appservices/search"> <search:result index="1" **uri="/example/11572018736249878991.json"** path="fn:doc("/example/11572018736249878991.json")" score="0" confidence="0" fitness="0" href="/v1/documents?uri=%2Fexample%2F11572018736249878991.json" mimetype="application/json" format="json"> <search:snippet/> </search:result> <search:result index="1" **uri="/example/5255172137182404803.xml"** path="fn:doc("/example/5255172137182404803.xml")" score="0" confidence="0" fitness="0" href="/v1/documents?uri=%2Fexample%2F5255172137182404803.xml" mimetype="application/xml" format="xml"> <search:snippet/> </search:result> </search:response>

JSON	{ "snippet-format": "empty-snippet",
	"total": 2,
	"start": 1,
	"page-length": 10,
	"results": [
	{
	"index": 1,
	"uri": "/example/13110416418064966920.json",
	"path": "fn:doc(\"/example/13110416418064966920.json\")",
	"score": 0,
	"confidence": 0,
	"fitness": 0,
	"href": "/v1/documents?uri=%2Fexample%2F13110416418064966920.json",
	"mimetype": "application/json",
	"format": "json"
	},
	{
	"index": 1,
	"uri": "/example/55172137182404803.xml",
	"path": "fn:doc(\"/example/55172137182404803.xml\")",
	"score": 0,
	"confidence": 0,
	"fitness": 0,
	"href": "/v1/documents?uri=%2Fexample%2F55172137182404803.xml",
	"mimetype": "application/xml",
	"format": "xml"
	}
	]
	}

5.2.10.5 Extending the Example

You can include more than one category of metadata in a metadata part. For example, if you change the metadata part contents in this example to the following, then the created documents are readable by users with the `readers` role and have the user-defined `source` property, as well as being in the "April 2014" collection.

Format	Metadata

XML	`<rapi:metadata xmlns:rapi="http://marklogic.com/rest-api">`
	`  <rapi:collections>`
	`  <rapi:collection>April 2014</rapi:collection>`
	`  </rapi:collections>`
	`  <rapi:permissions>`
	`  <rapi:permission>`
	`  <rapi:role-name>readers</rapi:role-name>`
	`  <rapi:capability>read</rapi:capability>`
	`  </rapi:permission>`
	`  </rapi:permissions>`
	`  <prop:properties xmlns:prop="http://marklogic.com/xdmp/property">`
	`  <source>wikipedia.org</source>`
	`  </prop:properties>`
	`</rapi:metadata>`

JSON	`{`
	`  "collections": [ "April 2014" ],`
	`  "permissions": [{`
	`  "role-name": "readers",`
	`  "capabilities": ["read"]`
	`  }],`
	`  "properties": {`
	`  "source": "wikipedia.org"`
	`  }`
	`}`

For more details about working with metadata using the REST API, see the following topics:

- "Working with Metadata" on page 124
- "Controlling Access to Documents Created with the REST API" on page 57

5.2.11 Generating Example Payloads with XQuery

This section describes how to generate a `multipart/mixed` payload using XQuery and QueryConsole. Each bulk write example includes a suitable XQuery query for generating its payload. If your development environment includes other HTTP client tools or libraries for payload generation, you can use those instead.

A multipart payload must contain CRLF control characters in specific places. For details, see RFC 1341, availble at the following URL:

http://www.w3.org/Protocols/rfc1341/7_2_Multipart.html

Cutting and pasting text from the documentation will not preserve these control characters. Editing a multipart body with many text editors also corrupts the control characters. Therefore, it is best to generate the example multipart bodies programmatically.

The procedure below uses `xdmp:multipart-encode` to create a multipart payload and `xdmp:save` to save the result to a file you can use as input to the `curl` command.

The `xdmp:multipart-encode` builtin function accepts a manifest and a sequence of nodes as input. The nodes are the part contents. The manifest contains the headers for each part. The calling sequence is:

```
xdmp:multpart-encode($part-boundary, $manifest, $part-contents)
```

The manifest must contain a `<part/>` element for each content part in the `$part-contents` sequence, in the same order as the content parts. See the example below.

Follow this procedure to write a `multipart/mixed` payload to a file. If you are not familiar with Query Console, see the *Query Console User's Guide*.

1. Open Query Console. For example, navigate to the following URL in your browser:

   `http://yourhost:8000/qconsole`

2. Create a new query by clicking on the "+" symbol at the top of the query editor.

3. Replace the default query contents with a body generation query. For example, replace the default query contents with the following query that generates 4 parts: JSON metadata, JSON content, XML metadata, and XML content.

```
xquery version "1.0-ml";

declare variable $OUTPUT-FILENAME := "/example/bulk-write-body";
declare variable $BOUNDARY := "BOUNDARY";

let $json-metadata1 := text { '{"quality" : 2 }'}
let $json-doc1 := text { '{ "key": "some json content" }' }
let $xml-metadata2 := document {
    <rapi:metadata xmlns:rapi="http://marklogic.com/rest-api">
      <rapi:quality>1</rapi:quality>
    </rapi:metadata>
  }
let $xml-doc2 := document{ <root>some xml content</root> }
let $manifest :=
  <manifest>
    <!-- JSON request default metadata -->
    <part>
      <headers>
         <Content-Type>application/json</Content-Type>
```

```
          <Content-Disposition>inline;
category=metadata</Content-Disposition>
      </headers>
    </part>
    <!-- JSON document -->
    <part>
       <headers>
          <Content-Type>application/json</Content-Type>
          <Content-Disposition>attachment;
filename="doc1.json"</Content-Disposition>
       </headers>
    </part>
    <!-- XML document-specific metadata -->
    <part>
      <headers>
        <Content-Type>application/xml</Content-Type>
        <Content-Disposition>attachment; filename="doc2.xml";
category=metadata</Content-Disposition>
      </headers>
    </part>
    <!-- XML content -->
    <part>
      <headers>
        <Content-Type>application/xml</Content-Type>
        <Content-Disposition>attachment;
filename="doc2.xml"</Content-Disposition>
      </headers>
    </part>
  </manifest>
return
xdmp:save($OUTPUT-FILENAME,
  xdmp:multipart-encode(
    $BOUNDARY, $manifest,
    ($json-metadata1,$json-doc1,$xml-metadata2,$xml-doc2)
  )
)
```

4. Change the `$OUTPUT-FILENAME` variable value to the full path where you want the payload to be saved. This directory must be writable by MarkLogic Server.

5. Click the Run button just below the query editor. The payload is saved to `$OUTPUT-FILENAME`.

If the output file is not created properly, check that the destination directory permissions permit MarkLogic Server to write a file to that location.

> **Note:** Depending on the editor you use, editing the saved file with a text editor can corrupt the payload by removing or converting the CRLF characters. Avoid editing the payload file unless you know your editor will preserve the CRLFs.

You can use the resulting output as input to the `curl` command using the `--data-binary` option. Use the same part boundary marker in your `curl` command that you used as the first parameter to `xdmp:http-encode` in the generation query.

For example, assuming you save the query output to `/example/bulk-write-body`, then the following command uses it as input to a bulk write request:

```
$ curl --anyauth --user user:password -X POST -i \
  --data-binary @/example/bulk-write-body \
  -H "Content-type: multipart/mixed; boundary=BOUNDARY"
  http://localhost:8003/v1/documents
```

5.3 Reading Multiple Documents by URI

You can retrieve multiple documents by URI in a single request by sending a POST request to the `/documents` service with multiple `uri` parameters and an Accept header of `multipart/mixed`. The URL should be of the following form:

```
http://host:port/version/documents?uri=uri-1&uri=uri-2&...
```

For example, the following command retrieves content for two documents, `aardvark.xml` and `camel.xml`:

```
$ curl --anyauth --user user:password -X GET -i \
  -H "Accept: multipart/mixed; boundary=BOUNDARY" \
  'http://localhost:8003/v1/documents?uri=aardvark.xml&uri=camel.xml'
```

MarkLogic Server returns a `multipart/mixed` response, with each part containing a requested document or its metadata. For details, see “Bulk Read Response Overview” on page 247.

The available request options are the same ones available when retrieving a single document and/or its metadata. For example:

- Use the `category` parameter to retrieve content, metadata (all or a subset), or both. For details, see “Retrieving Documents from the Database” on page 58.
- Use the `format` parameter to request metadata as either XML or JSON.
- Use the `transform` parameter to apply a read transform. For details, see “Transforming Content During Retrieval” on page 61.

  Note: Applying a transform creates an additional in-memory copy of each document, rather than streaming each document directly out of the database, so memory consumption is higher.

  Note: If you read both metadata and content for JSON documents, you should set format explicitly. Otherwise, the metadata format will vary depending upon whether your request includes one or multiple URIs.

5.4 Reading Multiple Documents Matching a Query

You can retrieve all documents that match a query by sending a GET or POST request to the `/search` or `/qbe` service with an Accept header value of `multipart/mixed`. You can retrieve content, metadata, or a combination of content and metadata for all matching documents.

- Overview of Bulk Read By Query
- Example: Using Query By Example (QBE)
- Example: Using a String, Structured, or Combined Query
- Including Search Results in the Response
- Paginating Results

5.4.1 Overview of Bulk Read By Query

To retrieve all documents from the database that match a query, use one of the following request methods, with the Accept header set to `multipart/mixed`:

Method	Description
`GET /version/search`	Retrieve documents matching a string and/or structured query with the query(s) in the `q` and/or `structuredQuery` request parameters.
`POST /version/search`	Retrieve documents matching a structured or combined query with the query in the POST body.
`GET /version/qbe`	Retrieve documents matching a Query By Example (QBE) with the query in the `query` request parameter.
`POST /version/qbe`	Retrieve documents matching a Query By Example (QBE) with the query in the POST body.

For example, the following command uses a string query to retrieve all documents containing the word bird:

```
$ curl --anyauth --user user:password -X GET -i \
  -H "Accept: multipart/mixed; boundary=BOUNDARY" \
  http://localhost:8003/v1/search?q=bird
```

MarkLogic Server returns a `multipart/mixed` response, with each part containing a matching document or its metadata. For details, see “Bulk Read Response Overview” on page 247.

For a more complete example, see “Example: Using a String, Structured, or Combined Query” on page 243.

Using a query to retrieve documents is the same as using `/search` and `/qbe` for a normal search, with the following exceptions:

- The Accept header must be `multipart/mixed`. This distinguishes your request from a normal search operation.
- No search response is included by default, but you can still request inclusion. For details, see "Including Search Results in the Response" on page 245.
- If you include a `transform` request parameter, the transform function is called on the returned documents and the search response but not on metadata. The transform must therefore be prepared to handle multiple kinds of input.
- Use the `category` parameter to specify the document data you want to retrieve: Content, metadata, or a metadata subset. You can retrieve a combination of these; for details, see "Bulk Read Response Overview" on page 247.
- Since a search response is not returned by default, the page range in the response is returned in vendor-specific response headers. For details, see "Paginating Results" on page 246.
- If there are no matches to the query, MarkLogic Server responds with status code 404 (Not Found), rather than a 200 (OK).

As with a normal search operation, you can include query options either by pre-installing them and naming them in the `options` request parameter, or by including them in a combined query in the POST body.

You can use a structured or combined query with the `/search` service to express complex queries. You can supply a structured query through the `structuredQuery` parameter on a GET request or in the body of a POST request. You can only supply a combined query in a POST request body.

See the following topics for more information on the query interfaces available through the REST API:

- "Querying Documents and Metadata" on page 132
- "Using Query By Example to Prototype a Query" on page 152
- "Specifying Dynamic Query Options with Combined Query" on page 161
- "Configuring Query Options" on page 167

5.4.2 Example: Using Query By Example (QBE)

This example demonstrates using a QBE to retrieve documents from the database using the `/qbe` service. You should be familiar with the basic `/qbe` REST API; for details, see "Using Query By Example to Prototype a Query" on page 152.

The following QBE matches documents with a kind XML element or JSON key with the value "bird":

Format	Query

XML	`<q:qbe xmlns:q="http://marklogic.com/appservices/querybyexample">`
	`  <q:query>`
	`  <kind>bird</kind>`
	`  </q:query>`
	`</q:qbe>`

JSON	`{ "$query":`
	`  { "kind": "bird" }`
	`}`

The following command uses the above query to retrieve all matching documents. Since the request does not include any category parameters, only document content is returned, one document per part. The number of documents matching the input query is returned in the `vnd.marklogic.result-estimate` response header. This number is equivalent to `@total` on a search response. The document URI, document type, and part contents are returned in the Content-Disposition header; for details, see "Bulk Read Response Overview" on page 247.

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X POST -d @query.xml -i \
  -H "Content-type: application/xml" \
  -H "Accept: multipart/mixed; boundary=BOUNDARY" \
  'http://localhost:8003/v1/qbe'
...
HTTP/1.1 200 OK
Content-type: multipart/mixed; boundary=BOUNDARY
vnd.marklogic.start: 1
vnd.marklogic.pageLength: 10
vnd.marklogic.result-estimate: 6
Server: MarkLogic
Content-Length: 4222
Connection: Keep-Alive
Keep-Alive: timeout=5

--BOUNDARY
Content-Type: application/xml
Content-Disposition: attachment; filename=/animals/vulture.xml;
category=content; format=xml
...
```

To make the equivalent request using a JSON QBE, change the Content-Type header to `application/json`. Note that the format of a QBE (XML or JSON) can affect the kinds of documents that match the query. For details, see Scoping a Search by Document Type in the *Search Developer's Guide*.

To return metadata as JSON rather than XML, use the `format` request parameter. For example, the following command returns document quality expressed as JSON:

```
$ curl --anyauth --user user:password -X POST -d @query.xml -i \
  -H "Content-type: application/xml" \
  -H "Accept: multipart/mixed; boundary=BOUNDARY" \
  'http://localhost:8003/v1/qbe?category=quality&format=json'
...
HTTP/1.1 200 OK
Content-type: multipart/mixed; boundary=BOUNDARY
vnd.marklogic.start: 1
vnd.marklogic.pageLength: 10
vnd.marklogic.result-estimate: 6
Server: MarkLogic
Content-Length: 4222
Connection: Keep-Alive
Keep-Alive: timeout=5

--BOUNDARY
Content-Type: application/json
Content-Disposition: attachment; filename=/animals/vulture.xml;
category=quality; format=json
...
```

If you use a GET request rather than a POST, you must URL-encode the `query` parameter value. For example, the following query uses the URL-encoded representation of the JSON query: `{ "$query": { "kind": "bird" } }`.

```
$ curl --anyauth --user user:password -X GET -i \
  -H "Accept: multipart/mixed; boundary=BOUNDARY" \
  'http://localhost:8004/v1/qbe?query=%7B%20%22%24query%22%3A%20%7B%20
%22kind%22%3A%20%22bird%22%20%7D%20%7D'
```

Use the `options` request parameter to include persistent query options in your request. Use the `transform` parameter to apply a read transform. For example:

```
$ curl --anyauth --user user:password -X POST -d @query.json -i \
  -H "Content-type: application/json" \
  -H "Accept: multipart/mixed; boundary=BOUNDARY" \
  'http://localhost:8003/v1/qbe?options=my-options&transform=my-enrich
ment
```

5.4.3 Example: Using a String, Structured, or Combined Query

This example demonstrates using a string, structured, or combined query to retrieve documents from the database using the `/search` service. The basic method is the same, no matter what query format you choose. You should be familiar with the basic `/search` REST API; for details, see “Querying Documents and Metadata” on page 132.

The following command uses a simple string query supplied through the `q` request parameter to retrieve all documents that contain "bird". Since the request does not include any `category` parameters, only document content is returned, one document per part. The number of documents matching the input query is returned in the `vnd.marklogic.result-estimate` response header. This number is equivalent to `@total` on a search response. The document URI, document type, and part contents are returned in the Content-Disposition header; for details, see "Bulk Read Response Overview" on page 247.

```
$ curl --anyauth --user user:password -X GET -i \
  -H "Accept: multipart/mixed; boundary=BOUNDARY" \
  http://localhost:8003/v1/search?q=bird
...
HTTP/1.1 200 OK
Content-type: multipart/mixed; boundary=BOUNDARY
vnd.marklogic.start: 1
vnd.marklogic.pageLength: 10
vnd.marklogic.result-estimate: 6
Server: MarkLogic
Content-Length: 1476
Connection: Keep-Alive
Keep-Alive: timeout=5

--BOUNDARY
Content-Type: text/xml
Content-Disposition: attachment; filename=/animals/vulture.xml;
category=content; format=xml
Content-Length: 93

<?xml version="1.0" encoding="UTF-8"?>
<animal><name>vulture</name><kind>bird</kind></animal>
...
```

To retrieve both documents and metadata, use the `category` parameter. The following example retrieves the document contents and the `quality` metadata. The response contains 2 parts for each matching document: A metadata part, immediately followed by the corresponding content part.

```
$ curl --anyauth --user user:password -X GET -i \
  -H "Accept: multipart/mixed; boundary=BOUNDARY" \
  'http://localhost:8003/v1/search?q=bird&category=content&category=quality'
...
HTTP/1.1 200 OK
Content-type: multipart/mixed; boundary=BOUNDARY
vnd.marklogic.start: 1
vnd.marklogic.pageLength: 10
vnd.marklogic.result-estimate: 6
Server: MarkLogic
Content-Length: 4222
Connection: Keep-Alive
Keep-Alive: timeout=5

--BOUNDARY
Content-Type: application/xml
```

```
Content-Disposition: attachment; filename=/animals/vulture.xml;
category=quality; format=xml
Content-Length: 299

?xml version="1.0" encoding="UTF-8"?>
<rapi:metadata uri="/animals/vulture.xml" ...>
  <rapi:quality>0</rapi:quality>
</rapi:metadata>
--BOUNDARY
Content-Type: text/xml
Content-Disposition: attachment; filename=/animals/vulture.xml;
category=content; format=xml
Content-Length: 93
...
```

To return the metadata as JSON rather than XML, add `format=json` to the request:

```
$ curl --anyauth --user user:password -X GET -i \
  -H "Accept: multipart/mixed; boundary=BOUNDARY" \
  'http://localhost:8003/v1/search?q=bird&category=content&category=qu
ality&format=json'
```

You can supply a structured query through the `structuredQuery` parameter on a GET request or in the body of a POST request. You can only supply a combined query in a POST request body. For example, assuming `query.json` contains a structured or combined query, the following command retrieves metadata for documents containing “bird”.

```
$ curl --anyauth --user user:password -X POST -d @./query.json -i
  -H "Content-Type: application/xml" \
  -H "Accept: multipart/mixed; boundary=BOUNDARY" \
  'http://localhost:8003/v1/search?category=metadata&format=json'
```

Use the `options` request parameter to include persistent query options in your request. Use the `transform` parameter to apply a read transform. For example:

```
$ curl --anyauth --user user:password -X GET -i \
  -H "Accept: multipart/mixed; boundary=BOUNDARY" \
  http://localhost:8003/v1/search?q=bird&options=my-options&transform=
my-enrichment
```

5.4.4 Including Search Results in the Response

By default, no search response is included in the request response when you perform a bulk read based on a query. To return search results in addition to matching documents or metadata, set the `view` request parameter.

For the complete set of values accepted by the `view` parameter, see the API reference for `GET:/v1/search`.

The search response is returned as XML by default. You can request a JSON search response by setting the `format` request parameter to `json`. If you request a JSON search response, your input query must also be JSON if it is not a string query. The `format` parameter also controls the format of returned metadata.

For example, the following command returns a search response, plus the documents matching the query:

```
$ curl --anyauth --user user:password -X GET -i \
  -H "Accept: multipart/mixed; boundary=BOUNDARY" \
  'http://localhost:8003/v1/search?q=bird&view=results'
...
HTTP/1.1 200 OK
...

--BOUNDARY
Content-Type: application/xml
Content-Disposition: inline
Content-Length: 3177

<?xml version="1.0" encoding="UTF-8"?>
<search:response snippet-format="snippet" total="6" ...>
...
</search:response>
--BOUNDARY
Content-Type: text/xml
Content-Disposition: attachment; filename=/animals/vulture.xml;
category=content; format=xml
Content-Length: 93

<?xml version="1.0" encoding="UTF-8"?>
<animal><name>vulture</name><kind>bird</kind></animal>
...
```

5.4.5 Paginating Results

As with a normal search operation, you can use the `start` and `pageLength` request parameters to retrieve results in batches. Use `start` to specify the index of the first result to return and `pageLength` to control the number results to return.

By default, queries return the first 10 matches. That is, the default start position is 1 and the default page length is 10. You can fetch successive results by incrementing the start position by the page length in each call.

In a normal search operation, the page range included in the response body is returned via the `start` and `page-length` attributes (or JSON key-value pairs) on the search response. Since a bulk read via query does not necessarily include a search response, the page range is returned in the vendor-specific response headers `vnd.marklogic.start` and `vnd.marklogic.pageLength`. If a search response is returned by your request, the `start` and `page-length` value in the search response match the header values.

Similarly, an estimate of the total number of matches is returned in the vendor-specific header `vnd.marklogic.result-estimate`. This is equivalent to the `total` value in a search response. As with any search, the actual number of matches might be different, depending upon whether you use filtered or unfiltered search.

For more information, see the Search Developer's Guide and Fast Pagination and Unfiltered Searches in the *Scalability, Availability, and Failover Guide*.

The following example command fetches the first 5 documents containing "bird". Notice that the response includes page range details in the `vnd.marklogic.*` headers.

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    'http://localhost:8003/v1/search?q=castle&start=1&pageLength=5'
...
HTTP/1.1 200 OK
Content-type: multipart/mixed; boundary=BOUNDARY
vnd.marklogic.start: 1
vnd.marklogic.pageLength: 5
vnd.marklogic.result-estimate: 16
Server: MarkLogic
Content-Length: 1225
Connection: Keep-Alive
Keep-Alive: timeout=5
...
```

To fetch the next set of matching documents, increment `start` by `pageLength` (5):

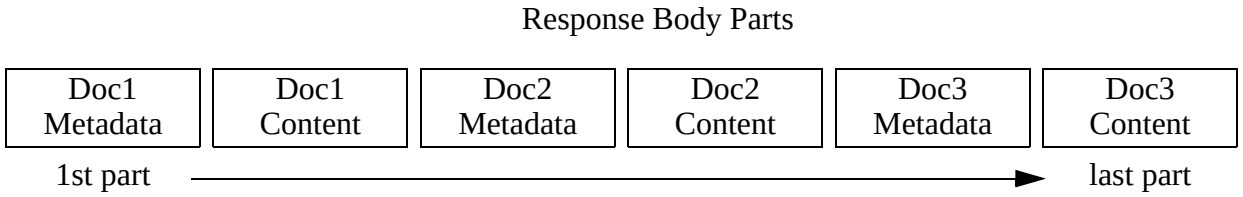
```
$ curl --anyauth --user user:password -X GET -i \
  -H "Accept: multipart/mixed; boundary=BOUNDARY" \
  'http://localhost:8003/v1/search?q=bird&start=6&pageLength=5'
...
HTTP/1.1 200 OK
Content-type: multipart/mixed; boundary=BOUNDARY
vnd.marklogic.start: 6
vnd.marklogic.pageLength: 5
vnd.marklogic.result-estimate: 16
Server: MarkLogic
Content-Length: 1225
Connection: Keep-Alive
Keep-Alive: timeout=5
...
```

When the page length returned in `vnd.marklogic.pageLength` is less than the request `pageLength`, no more matching documents are available.

5.5 Bulk Read Response Overview

When you retrieve multiple documents in a single request using a list of URIs or a query, the results are returned in a `multipart/mixed` response that contains a part for each returned document or document metadata.

If you request both content and metadata, the content and metadata are returned in separate parts, with the metadata part immediately preceding the content part. If you request multiple metadata sub-categories, such as `quality` and `permissions`, then each metadata part contains all the requested subcategory values. The diagram below illustrates the response part stream for a request the returns both content and metadata.



When you retrieve multiple documents using a query, the response can also include the search response as the first part; for details, see “Including Search Results in the Response” on page 245.

The part Content-Type header contains the MIME type of the data in the part body. The Content-Type of a part is determined as follows:

- For a content part, the MIME type is determined by the MarkLogic Server MIME type mapping corresponding to the document URI extension. To review or change the MIME type mapping, see the “Mimetypes” subsection of the Groups section of the Admin Interface.
- For a metadata part, the MIME type is always `application/xml` or `application/json`. The default format is XML. Use the `format` request parameter to control the metadata MIME type.
- If a search response is included in the response, its MIME type is XML unless you use the `format=json` request parameter. For details, see “Including Search Results in the Response” on page 245.

The Content-Disposition header contains the source document URI, the kind of data in the part, and format of the data. The Content-Disposition header for a part has the following form:

```
Content-Disposition: attachment;filename=doc-uri;
category=data-category;...format=data-format
```

Where *`doc-uri`* is the database URI of the document from which the content or metadata in the part was extracted, and *`data-category`* is either `content`, `metadata`, or a specific metadata sub-category such as `permissions` or `quality`. For metadata, `format` is always `xml` or `json`. For content, `format` corresponds to the database document type: `xml`, `json`, `text`, or `binary`.

If optimistic locking or content versioning is enabled, the Content-Disposition header for a content part also contains a document version id of the form `versionId=`*`id`*; for details, see “Using Optimistic Locking to Update Documents” on page 114.

For example, the following bulk read request retrieves content, permissions, and document quality, so a content and a metadata part is returned for each document. The Content-Disposition header for each metadata part includes both `category=permissions` and `category=quality`.

```
$ curl --anyauth --user user:password -X GET -i \
  -H "Accept: multipart/mixed; boundary=BOUNDARY" \
  'http://localhost:8003/v1/documents?category=content&category=permis
sions&category=quality&uri=...'

--BOUNDARY
Content-Type: application/xml
Content-Disposition: attachment; filename=...; category=permissions;
category=quality; format=xml
...
--BOUNDARY
Content-Type: text/xml
Content-Disposition: attachment; filename=...; category=content;
format=xml
...
```

The table below contains additional examples of returned part headers.

Part Description	Example Response Part Headers

XML content	`Content-type: application/xml`
	`Content-Disposition: attachment; filename=some.xml;`
	`category=content; format=xml`

JSON content	`Content-type: application/json`
	`Content-Disposition: attachment;filename=some.json;`
	`category=content; format=json`

Binary content	`Content-type: image/jpeg`
	`Content-Disposition: attachment;filename=some.jpg;`
	`category=content; format=binary`

All metadata for a document, expressed as XML	`Content-type: application/xml`
	`Content-Disposition: attachment; filename=some.xml;`
	`category=metadata; format=xml`

Selected metadata, expressed as json	`Content-type: application/json`
	`Content-Disposition: attachment; filename=some.json;`
	`category=quality; category=properties;`
	`format=json`

6.0 Managing Transactions

The MarkLogic REST API provides transaction control through the `/transactions` service and a `txid` parameter available on document manipulation and search requests. This section covers the following topics:

- Service Summary
- Overview of RESTful Transactions
- Creating a Transaction
- Associating a Transaction with a Request
- Committing or Rolling Back a Transaction
- Checking Transaction Status
- Managing Transactions When Using a Load Balancer

6.1 Service Summary

Many of the services available through the MarkLogic REST API allow you to perform an operation in the context of a specific transaction by passing a transaction id in a `txid` parameter. The `/transactions` service supports the creation and administration of these transactions.

The `/transactions` service supports the following operations:

Operation	Method	Description	More Information
Create	POST	Create a multi-statement transaction. See “Creating a Transaction” on page 252.	“Creating a Transaction” on page 252
Commit	PUT	Commit an open transaction.	“Committing or Rolling Back a Transaction” on page 253
Rollback	DELETE	Rollback a transaction, discarding all related changes.	“Committing or Rolling Back a Transaction” on page 253
Status	GET	Retrieve transaction status in XML or JSON.	“Checking Transaction Status” on page 253

6.2 Overview of RESTful Transactions

This section gives a brief introduction to the MarkLogic Server transaction model as it applies to the MarkLogic REST API. For a full discussion of the transaction model, see Understanding Transactions in MarkLogic Server in the *Application Developer's Guide*..

By default, each request to a MarkLogic REST API service is equivalent to a single statement transaction. That is, the request is evaluated as single transaction.

For example, when you update a document in the database by making a PUT request to the `/documents` service, the service effectively creates a new transaction, updates the document, commits the transaction, and then sends back a response. The update is immediately visible in the database and available to other requests.

The MarkLogic REST API also allows your application to take direct control of transaction boundaries so that multiple request are evaluated in the same transaction context. This is equivalent to the multi-statement transactions described in Multi-Statement Transaction Concept Summary in the *Application Developer's Guide*.

Using multi-statement transactions, you can make several update requests and commit them as a single transaction, ensuring either all or none of the updates appear in the database. The document manipulation and search capabilities of the MarkLogic REST API support multi-statement transactions.

To use multi-statement transactions:

1. Create a multi-statement transaction by sending a POST request to the `/transactions` service. The service returns a transaction id. See "Creating a Transaction" on page 252.

2. Evaluate one or more requests in the context of the transaction by including the transaction id in the `txid` request parameter. See "Associating a Transaction with a Request" on page 253.

3. Commit the transaction by sending a PUT request to the `/transactions` service, or rollback the transaction by sending a DELETE request to the `/transactions` service. See "Committing or Rolling Back a Transaction" on page 253.

If your application interacts with MarkLogic Server through a load balancer, you might need to include a HostId cookie in your requests to preserve session affinity. For details, see "Managing Transactions When Using a Load Balancer" on page 255.

When you explicitly create a transaction, you must explicitly commit or roll it back. Failure to do so leaves the transaction open until the request or transaction timeout expires. Open transactions can hold locks and consume system resources.

If the request or transaction timeout expires before a transaction is committed, the transaction is automatically rolled back and all updates are discarded. Configure the request timeout of the App Server using the Admin UI. Configure the timeout of a single transaction by setting the `timeLimit` request parameter during transaction creation.

6.3 Creating a Transaction

To create a transaction, send a POST request to the `/transactions` service with a URL of the form:

```
http://host:port/version/transactions
```

The service responds with a transaction id in the `Location` response header. Use the transaction id to control the transaction in which MarkLogic Server evaluates future requests; see “Associating a Transaction with a Request” on page 253.

The transaction id returned in the Location header is of the form:

```
/transactions/hostid_txnid
```

For example:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl -X POST -d "" --anyauth --user user:password \
    http://localhost:8003/v1/transactions
...
HTTP/1.1 303 See Created Transaction
Location: /transactions/9146583258914468644_3148548124558550433
```

When supplying the transaction id to future requests, use only the `hostid_txnid` portion. For example:

```
$ curl -X PUT -T ./example.xml --anyauth --user user:password \
  'http://localhost:8003/v1/documents?uri=/xml/example.xml&txid=914658
3258914468644_3148548124558550433'
```

Transactions created using the /transactions service must be explicitly committed or rolled back. Failure to commit or rollback the transaction before the request timeout expires causes an automatic rollback. You can assign a shorter time limit to a transaction using the timeLimit URL parameter:

```
http://host:port/version/transactions?timeLimit=seconds
```

As a convenience for identifying a transaction in server status reports, you can assign a name to the transaction. The default transaction name is “client-txn”. To assign a transaction, add the `name` URL parameter to your request:

```
http://host:port/version/transactions?name=txn_name
```

6.4 Associating a Transaction with a Request

Once you create a transaction, as described in “Creating a Transaction” on page 252, you can evaluate document manipulation and search requests in that transaction by supplying the transaction id wherever a `txid` URL parameter is supported.

For example, to perform a document update in the context of a specific transaction, include the transaction id in a PUT request to the `/documents` service:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl -X PUT -T ./example.xml --anyauth --user user:password \
  'http://localhost:8003/v1/documents?uri=/xml/example.xml&txid=914658
3258914468644_3148548124558550433'
```

Updates associated with an explicit transaction are visible to other requests executed within the same transaction, but they are not visible outside the transaction until the transaction is committed. See “Committing or Rolling Back a Transaction” on page 253.

6.5 Committing or Rolling Back a Transaction

To commit a transaction created by making a POST request to the `/transactions` service, send a POST request to the `/transactions/{txid}` service with a URL of the form:

```
http://host:port/version/transactions/txid?result=outcome
```

Where *`txid`* is a transaction id of the form *`hostid_txnid`*, as returned in the `Location` header of the transaction creation request response, and *`outcome`* is either `commit` or `rollback`.

MarkLogic Server responds with a 204 when a transaction is successfully committed or rolled back, or if the transaction no longer exists.

Transactions created using the `/transactions` service should be explicitly commited or rolled back. Failure to commit or rollback the transaction before the request timeout expires causes an automatic rollback. Leaving transactions open unnecessarily potentially holds locks on documents and consume system resources.

6.6 Checking Transaction Status

To query the status of a transaction, send a GET request to the `/transactions` service with a URL of the form:

```
http://host:port/version/transactions/txid
```

Where `txid` is a transaction id of the form *`hostid_txnid`*, as returned in the `Location` header of the transaction creation request response.

MarkLogic Server responds with transaction status information in XML or JSON, depending upon the format requested through the HTTP `Accept` header or the `format` URL `parameter`. The default format is XML. If both the `Accept` header and the `format` parameter are set, the format parameter takes precedence.

To request a specific format, set the `Accept` header to `application/json` or `application/xml`, or set the `format` URL parameter to `xml` or `json`. The following example requests JSON output using the `Accept` header:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl -X GET --anyauth --user user:password \
    -H "Accept: application/json" \
    http://localhost:8003/v1/transactions/1506593501555860493_21610463
5458437451
```

To make the equivalent request using the `format` parameter:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl -X GET --anyauth --user user:password \
    http://localhost:8003/v1/transactions/1506593501555860493_21610463
5458437451?format=json
```

The XML status information returned in the response takes the following form:

```
<rapi:transaction-status xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:host>
    <rapi:host-id>1506593501555860493</rapi:host-id>
    <rapi:host-name>kim-3.marklogic.com</rapi:host-name>
  </rapi:host>
  <rapi:server>
    <rapi:server-id>17994162681305741146</rapi:server-id>
    <rapi:server-name>RESTstop</rapi:server-name>
  </rapi:server>
  <rapi:database>
    <rapi:database-id>8896678293064963766</rapi:database-id>
    <rapi:database-name>bill</rapi:database-name>
  </rapi:database>
  <rapi:transaction-id>216104635458437451</rapi:transaction-id>
  <rapi:transaction-name>client-txn</rapi:transaction-name>
  <rapi:transaction-mode>update</rapi:transaction-mode>
  <rapi:transaction-timestamp>0</rapi:transaction-timestamp>
  <rapi:transaction-state>idle</rapi:transaction-state>
  <rapi:canceled>false</rapi:canceled>
  <rapi:start-time>2012-08-18T17:23:00-07:00</rapi:start-time>
  <rapi:time-limit>600</rapi:time-limit>
  <rapi:max-time-limit>3600</rapi:max-time-limit>
  <rapi:user>7071164303237443533</rapi:user>
  <rapi:admin>true</rapi:admin>
</rapi:transaction-status>
```

The JSON status information returned in the response takes the following form:

```
{
  "rapi:transaction-status": {
    "rapi:host": {
      "rapi:host-id": "15065935015558604 93",
      "rapi:host-name": "kim-3.marklogic.com"
    },
    "rapi:server": {
      "rapi:server-id": "17994162681305741146",
      "rapi:server-name": "RESTstop"
    },
    "rapi:database": {
      "rapi:database-id": "8896678293064963766",
      "rapi:database-name": "bill"
    },
    "rapi:transaction-id": "216104635458437451",
    "rapi:transaction-name": "client-txn",
    "rapi:transaction-mode": "update",
    "rapi:transaction-timestamp": "0",
    "rapi:transaction-state": "idle",
    "rapi:canceled": "false",
    "rapi:start-time": "2012-08-18T17:23:00-07:00",
    "rapi:time-limit": "600",
    "rapi:max-time-limit": "3600",
    "rapi:user": "7071164303237443533",
    "rapi:admin": "true"
  }
}
```

You can also check transaction status using the Admin Interface or `xdmp:host-status`.

6.7 Managing Transactions When Using a Load Balancer

This section applies only to client applications that use multi-statement transactions and interact with a MarkLogic Server cluster through a load balancer. Requests that include a transaction id (`txid`) request parameter are part of a multi-statement transaction.

When you use a load balancer, it is possible for requests from your application to MarkLogic Server to be routed to different hosts, even within the same session. This has no effect on most interactions with MarkLogic Server, but all requests that are part of the same multi-statement transaction must be routed to the same host within your MarkLogic cluster. This consistent routing through a load balancer is called *session affinity.*

To support session affinity, the REST API sets a `HostId` cookie on any response that creates or does work in the context of a multi-statement transaction. The cookie value is an unsigned long representation of the host identifier and does not include any session state.

The REST API supports session affinity through a `HostId` cookie, as follows:

- When you create a transaction by making a POST request to the `/transactions` service, the response includes a transaction id in the Location header and a `HostId` cookie in the Set-cookie header.
- When you use the same transaction id in subsequent requests, you should also include the same `HostId` cookie in the request headers.
- If you configure your load balancer to use the `HostId` cookie for session affinity, all requests associated with the same transaction id will be routed to the same host.
- When you commit or rollback an open transaction, you can discard the associated `HostId` cookie.

Most client applications do not need to persist the cookie or arrange for its expiration. The cookie does not include any session state. The cookie value is not used by the REST API.

The exact steps required to configure session affinity in the load balancer so depend on the load balancer. Consult your load balancer documentation for details.

7.0 Alerting

The following topics are covered:

- Summary of /alert Services
- Alerting Pre-Requisites
- Alerting Concepts
- Defining an Alerting Rule
- Installing an Alerting Rule
- Testing for Matches to Alerting Rules
- Retrieving Rule Definitions
- Testing for the Existence of Rule
- Deleting a Rule

7.1 Summary of /alert Services

The table below provides a brief summary of the alerting services provided by MarkLogic REST API:

Operation	Methods	Description
`/alert/rules`	`GET`	Retrieve definitions of all installed alerting rules.

`/alert/rules/{name}`	`HEAD`
	`GET`
	`PUT`
	`DELETE`
		Maintain alerting rules. You can create/modify, delete, and test for the existence of rules by name. Rules can be expressed in XML or JSON.

`/alert/match`	`GET` `POST`	Test one or more documents for matches against all installed rules or a specified subset of rules. The input documents can be specified using a query, URIs, or by passing the document content into the request. You can define an XQuery function to transform the match results in an application-specific way.

7.2 Alerting Pre-Requisites

This feature depends internally upon the XQuery Alerting API and reverse query indexing. As such, you must have a valid alerting license key to use this feature.

You should also enable “fast reverse searches” on the content database associated with your REST API instance. Enable fast reverse searches using the Admin Interface, as described in Indexes for Reverse Queries in *Search Developer's Guide*, or using the XQuery function `admin:database-set-fast-reverse-searches`.

The `rest-writer` role or equivalent privileges is required to create and delete alerting rules. All other operations require the `rest-reader` or equivalent privileges. Additional privileges may be required to access input documents during rule match operations.

7.3 Alerting Concepts

An *alerting application* is one that takes action whenever content matches a pre-defined set of criteria. For example, send an email notification to a user whenever a document about influenza is added to the database. In this case, the criteria might be “the abstract contains the word influenza”, and the action is “send an email”.

MarkLogic Server supports server-side alerting through the XQuery API and Content Processing Framework (CPF), and client-side alerting through the REST and Java APIs.

A server-side alerting application usually uses a “push” model. You register alerting rules and XQuery action functions with MarkLogic Server. Whenever content matches the rules, MarkLogic Server evaluates the action functions. For details, see Creating Alerting Applications in *Search Developer's Guide*.

By contrast, a client-side alerting application uses a “pull” alerting model. You register altering rules with MarkLogic Server, as in the push model. However, your application must poll MarkLogic Server for matches to the configured rules, and the application initiates actions in response to matches. This is the model used by the MarkLogic REST API.

An *alerting rule* is a query used in a reverse query to determine whether or not a search using that query would match a given document. A normal search query asks “What documents match these search criteria?” A reverse query asks “What criteria match this document?” In the influenza example above, you might define a rule that is a word query for “influenza”, with an element constraint of `<abstract/>`. Queries associated with alerting rules are stored in the database.

MarkLogic Server provides fast, scalable rule matching by storing queries in alerting rules in the database and indexing them in the reverse query index. You must explicitly enable “fast reverse searches” on your content database to take advantage of the reverse quer index. For details, see Indexes for Reverse Queries in *Search Developer's Guide*.

> **Note:** A valid alerting license key is required to use the reverse index and the REST API alerting features.

Use the procedures described in this chapter to create and maintain search rules and to test documents for matches to the rules installed in your REST API instance. Determining what actions to take in response to a match and initiating those actions is left to the application.

7.4 Defining an Alerting Rule

An alerting rule is defined by a name, a query, and optional metadata. The core of a rule is the combined query that describes the search criteria to use in future match operation. A combined query is a `search` wrapper around a string and/or structured query plus query options; for details, see "Specifying Dynamic Query Options with Combined Query" on page 161.

All rules must have a name. You can either specify the name explicitly in the rule definition, or have MarkLogic Server internally add a name derived from the *`name`* path step in the URI used to install the rule in the REST API instance. If you specify an explicit name in the definition, it must match the *`name`* in the PUT request URI.

Use the following template for defining the rule in the request body.

Format	Alerting RuleTemplate

XML	`<rapi:rule xmlns:rapi="http://marklogic.com/rest-api">`
	`  <rapi:name>`*`rule-name`*`</rapi:name>`
	`  <rapi:description>`*`optional description`*`</rapi:description">`
	`  <rapi:rule-metadata>`
	`  `*`optional user-defined metadata`*
	`  </rapi:rule-metadata>`
	`  <!--`*` required combined query `*`-->`
	`</rapi:rule>`

JSON	`{"rule":`
	`  "name": "`*`rule-name`*`",`
	`  "description": "`*`optional description`*`",`
	`  "rule-metadata":`*` optional user-defined metadata`*`,`
	`  "search" :`*` a combined query`*
	`}`

For example, the following rule enables you to test whether a document or a set of documents matches the string query "xdmp" in a case-sensitive search.

```
<rapi:rule xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:name>Example Rule</rapi:name>
  <search:search
      xmlns:search="http://marklogic.com/appservices/search">
    <search:qtext>xdmp</search:qtext>
    <search:options>
      <search:term>
        <search:term-option>case-sensitive</search:term-option>
      </search:term>
    </search:options>
  </search:search>
</rapi:rule>
```

You can include an optional description and metadata. This data is not used by MarkLogic Server, but it is returned in match results and when you fetch back a rule definition. The structure of the rule metadata is user-defined. The following example rule adds a description and an author to the above rule:

```
<rapi:rule xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:name>Example Rule</rapi:name>
  <search:search
      xmlns:search="http://marklogic.com/appservices/search">
    <search:qtext>xdmp</search:qtext>
    <search:options>
      <search:term>
        <search:term-option>case-sensitive</search:term-option>
      </search:term>
    </search:options>
  </search:search>
  <rapi:description>An example rule.</rapi:description>
  <rapi:rule-metatdata>
    <author>me</author>
  </rapi:rule-metadata>
</rapi:rule>
```

The following example shows an equivalent rule expressed as JSON.

```
{ "rule": {
  "name" : "json-example",
  "search" : {
    "qtext" : "xdmp",
    "options" : {
      "term" : { "term-option" : "case-sensitive" }
    }
  },
  "description": "A JSON example rule.",
  "rule-metadata" : { "author" : "me" }
}}
```

Note: If you insert metadata as JSON and retrieve it as XML, reference the XML metadata elements in the namespace `http://marklogic.com/rest-api`.

7.5 Installing an Alerting Rule

Before you can perform a match operation against an alerting rule, you must install the rule in your REST API instance. To persist a rule in your REST API instance, send a PUT request to the `/alert/rules/{name}` service with a URL of the following form. The request body must contain an XML or JSON rule definition of the form described in “Defining an Alerting Rule” on page 259.

```
http://host:port/version/alert/rules/name
```

Where *name* is the name of the rule. If you specify an explicit name in the rule definition, it must match `name` in the PUT request URI. If you do not specify an explicit name in the rule definition, MarkLogic Server adds a name corresponding to the `name` in the request URI.

The following example command installs a rule under the name “example”, assuming the file `rule.xml` contains the rule definition:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT -d @./rule.xml \
    -i -H "Content-type: application/xml" \
    'http://localhost:8003/v1/alert/rules/example'
```

To install a rule expressed as JSON, set the Content-type header to `application/json` or set the format request parameter to `json`. For example:

```
$ curl --anyauth --user user:password -X PUT -d @./rule.json \
    -i -H "Content-type: application/json" \
    'http://localhost:8003/v1/alert/rules/json-example'
```

7.6 Testing for Matches to Alerting Rules

Once you install alerting rules in your REST API instance, use the `/alert/match` service to determine which rules match one or more input documents. You can select the input documents using a database query or database URIs, or by passing a transient document.

This section covers the following topics:

- Basic Steps
- Identifying Input Documents Using a Query
- Identifying Input Documents Using URIs
- Matching Against a Transient Document
- Filtering Match Results
- Transforming Match Results

7.6.1 Basic Steps

To test one or more documents for a match against the rules, send a GET or POST request to the `/alert/match` service with a URL of the following form:

```
http://host:port/version/alert/match
```

The following example command returns all rules that match the document with the URI `/example/doc.xml`:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
```

```
-i -H "Accept: application/xml" \
'http://localhost:8003/v1/alert/match?uri=/example/doc.xml'
```

You must use one of the following methods to select the document(s) to test for a match:

- Query: Use a string, structured, or combined query to select documents in the database. For details, see “Identifying Input Documents Using a Query” on page 263. Use one of the following forms of request:

  ```
  GET
  http://host:port/version/alert/match?q=q1&structuredQuery=q2&options=q
  uery-options

  POST http://host:port/version/alert/match
  (post body contains a combined query)
  ```

- URI: Use one or more document URIs to select documents in the database. For details, see “Identifying Input Documents Using URIs” on page 264. Use a request of the following form:

  ```
  GET http://host:port/version/alert/match?uri=document-uri
  ```

- Content: Supply a transient document in the request body instead of using documents in the database. For details, see “Matching Against a Transient Document” on page 264. Use a request of the following form:

  ```
  POST http://host:port/version/alert/match
  (request body contains an XML document)
  ```

The response is a `rules` object that contains the definition of every rule that matches at least one of the input documents. You can request output in either XML or JSON using the HTTP Accept header. The data in the response body has the following form:

Format	Alerting RuleTemplate

XML	`<rapi:rules xmlns:rapi="http://marklogic.com/rest-api">`
	`  <rapi:rule>`*rule definition*`</rapi:rule>`
	`  <rapi:rule>`*rule definition*`</rapi:rule>`
	`</rapi:rules>`

JSON	`{"rules": [`
	`  {"rule": {` *rule definition* `},`
	`  {"rule": {` *rule definition* `}`
	`] }`

For the structure of a rule definition, see “Defining an Alerting Rule” on page 259.

The results do not indicate which input documents match each rule. If you need to know this, you should match against one document at a time.

7.6.2 Identifying Input Documents Using a Query

To use a query to select the documents to test for rule matches, do one of the following:

- Make a GET request that includes a string query in the `q` request parameter and/or a structured query in the `structuredQuery` request parameter. If you include both, the two queries are AND'd together.
- Make a POST request that includes a combined query in the request body.

The syntax and semantics of the queries are the same as when making a GET or POST request to the `/search` service. For details, see "Querying Documents and Metadata" on page 132 and "Specifying Dynamic Query Options with Combined Query" on page 161.

The following example GET request uses a string query to select all documents that contain the search terms "cat" and "dog". The search is performed using the persistent query options stored under the name "my-options".

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    -i -H "Accept: application/xml" \
    'http://localhost:8003/v1/alert/match?options=my-options&q=cat AND
dog'
```

The following example POST request uses a combined query to perform a similar match. Note that query options can be defined at request time when you use a combined query.

```
$ cat match-body.xml
<search xmlns="http://marklogic.com/appservices/search">
  <qtext>xdmp</qtext>
  <options>
    <term>
      <term-option>case-sensitive</term-option>
    </term>
  </options>
</search>

$ curl --anyauth --user user:password -X POST -d @./match-body.xml \
    -i -H "Content-type: application/xml" \
    http://localhost:8003/v1/alert/match
```

To express the combined query in JSON, use a query and command similar to the following:

```
$ cat match-body.json
{ "search" : {
    "qtext" : "xdmp",
    "options" : {
      "term" : { "term-option" : "case-sensitive" }
```

```
    }
}}

$ curl --anyauth --user user:password -X POST -d @./match-body.json \
    -i -H "Content-type: application/json" \
    http://localhost:8003/v1/alert/match
```

7.6.3 Identifying Input Documents Using URIs

To specify the input documents for an alert match using URIs, include one or more `uri` request parameters in a GET request to `/alert/match`. Each URI must identify a document, not a database directory.

The following example command tests the documents `/example/doc1.xml` and `/example/doc2.xml` for rule matches.

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    -i -H "Accept: application/xml" \
    'http://localhost:8003/v1/alert/match?uri=/example/doc1.xml&uri=/e
xample/doc2.xml'
```

7.6.4 Matching Against a Transient Document

You can test for rules matches against a document that is not stored in the database by making a POST request to `/alert/match` with the document in the request body.

The following example command tests which rules match the content in the local file `transient.xml`:

```
$ cat > transient.xml
<function>
  <prefix>xdmp</prefix>
  <name>document-delete</name>
</function>

# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X POST \
    -i -H "Content-type: application/xml" -d @./transient.xml \
    'http://localhost:8003/v1/alert/match'
```

You can use either an XML or JSON document as input. The following example command use JSON input:

```
$ curl --anyauth --user user:password -X POST \
    -i -H "Content-type: application/json" -d @./transient.json \
    'http://localhost:8003/v1/alert/match'
```

7.6.5 Filtering Match Results

By default, the response to an alert match includes all matching rules. Use the `rule` request parameter to limit the results to a subset of the matching rules. For example, the response to the following command will include at most the definition of the rules named “one” and “two”, even if more rules match the input document.

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    -i -H "Accept: application/xml" \
    'http://localhost:8003/v1/alert/match?uri=/example/doc.xml&rule=on
e&rule=two'
...
<rapi:rules xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:rule>
    <rapi:name>one</rapi:name>
    <rapi:description>Rule 1</rapi:description>
    <search:search
        xmlns:search="http://marklogic.com/appservices/search">
      <search:qtext>xdmp</search:qtext>
    </search:search>
  </rapi:rule>
</rapi/rules>
```

The following is the equivalent command using JSON output.

```
$ curl --anyauth --user user:password -X GET \
    -i -H "Accept: application/json" \
    'http://localhost:8003/v1/alert/match?uri=/example/doc.xml&rule=on
e&rule=two'
...
{ "rules": [
    { "rule": {
        "name": "one",
        "description": "Rule 1",
        "search": { "qtext": ["xdmp"] }
    } }
] }
```

7.6.6 Transforming Match Results

You can make arbitrary changes to the response from match request by applying a user-defined XQuery transformation function to the response. This section covers the following topics:

- Writing a Match Result Transform
- Using a Match Result Transform

7.6.6.1 Writing a Match Result Transform

Alert match transforms use the same interface and framework as content transformations applied during document ingestion, described in “Working With Content Transformations” on page 269.

Your transform function receives the raw XML match result data as input, such as a document with a <rapid:rules/> root element. For example:

```
<rapi:rules xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:rule>
    <rapi:name>one</rapi:name>
    <rapi:description>Rule 1</rapi:description>
    <search:search
        xmlns:search="http://marklogic.com/appservices/search">
      <search:qtext>xdmp</search:qtext>
    </search:search>
  </rapi:rule>
</rapi/rules>
```

If your function produces XML output and the client application requested JSON output, MarkLogic Server will transform your output to JSON only if one of the following conditions are met.

- Your function produces an XML document that conforms to the "normal" output from the search operation. For example, a document with a `<rapi:rules/>` root element whose contents are changed in a way that preserves the normal structure.
- Your function produces an XML document with a root element in the namespace `http://marklogic.com/xdmp/json/basic` that can be transformed by `json:transform-to-json`.

Under all other circumstances, the response body contains exactly the output returned by your transform function.

7.6.6.2 Using a Match Result Transform

Follow this procedure to apply a transformation:

1. Create a transform function according to the interface described in "Writing Transformations" on page 269.

2. Install your transform function on the REST API instance following the instructions in "Installing Transformations" on page 277.

3. Apply your transform to a request to the `/alert/match` service by specifying the name in the `transform` request parameter, as described in "Applying Transformations" on page 278.

If the transform function expects additional parameters, specify them by name using `trans:{name}` request parameters.

The following example command performs an alert match using the document with URI `/example/doc.xml` as input. The match results are transformed by calling the XQuery transform function installed with the name `my-alert-txfm`. A single parameter named `my-param` is passed into the transform function.

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    -i -H "Accept: application/xml" \
    'http://localhost:8003/v1/alert/match?uri=/example/doc.xml&transfo
rm=my-alert-txfm&trans:my-param=value'
```

7.7 Retrieving Rule Definitions

You can retrieve the definition of all installed rules or the definition of a single installed rule identified by name.

To retrieve the definitions of all installed rules, send a GET request to the `/alert/rules` service with a URL of the following form:

```
http://host:port/version/alert/rules
```

To retrieve the definition of a single rule by name, send a GET request to the `/alert/rules/{name}` service with a URL of the following form:

```
http://host:port/version/alert/rules/{name}
```

Use the HTTP Accept header or the `format` request parameter to request results in either XML or JSON.

The following example command returns the definition of the rule named “Rule1”, in XML:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    -i -H "Accept: application/xml" \
    'http://localhost:8003/v1/alert/rules/Rule1'

<rapi:rule xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:name>Rule1</rapi:name>
  <search:search
      xmlns:search="http://marklogic.com/appservices/search">
    <search:qtext>xdmp</search:qtext>
  </search:search>
</rapi:rule>
```

7.8 Testing for the Existence of Rule

To test for the existence of a particular rule without fetching back the defining, send a HEAD request to the `/alert/rules/{name}` service with a URL of the following form:

```
http://host:port/version/alert/rules/{name}
```

If the named rule exists, MarkLogic Server responds with status code 200 (OK). If no such rule exists, MarkLogic Server responds with status 404 (Not Found).

7.9 Deleting a Rule

To delete a previously installed rule, send a DELETE request to the `/alert/rules/{name}` service with a URL of the following form:

```
http://host:port/version/alert/rules/{name}
```

8.0 Working With Content Transformations

The MarkLogic REST API allows you to create custom content transformations and apply them during operations such as document ingestion and search. Transforms can be implemented as XQuery library modules or XSLT stylesheets, and can accept transform-specific parameters. Transforms can be applied on any MarkLogic REST API service that accepts a `transform` parameter.

This chapter covers the following topics:

- Writing Transformations
- Installing Transformations
- Applying Transformations
- Discovering Transformations
- Retrieving the Implementation of a Transformation
- XQuery Example: Adding an Attribute During Ingestion
- XSLT Example: Adding an Attribute During Ingestion

8.1 Writing Transformations

This section covers the following topics:

- Guidelines for Writing Transforms
- Writing XQuery Transformations
- Writing XSLT Transformations
- Expected Input and Output
- Reporting Errors
- Context Map Keys
- Controlling Transaction Mode

8.1.1 Guidelines for Writing Transforms

Keep the following guidelines in mind when authoring a transform:

- A transform is one step in a document processing pipeline controlled by the MarkLogic REST API framework. As such, transforms typically should not have side-effects.
- A transform operates on an in-memory input document. MarkLogic Server builtin functions that operate on in-database documents, such as `xdmp:node-replace`, cannot be applied directly to the input document. To modify the structure of the input document, perform the modifications during construction of a copy of the input content.

- A transform executes in the transaction context of the HTTP request on whose behalf it is called.
- A transform should not alter the HTTP request context. For example, you should not set the response code or headers directly. The MarkLogic REST API framework manages the request context.
- When working with JSON, your transform always receives XML input.
- MarkLogic Server automatically applies an XML to JSON transformation on the output of your function if certain requirements are met and the client requested JSON output. For details, see “Expected Input and Output” on page 273.
- Report errors by throwing one of the exceptions provided by the MarkLogic REST API. For details, see “Reporting Errors” on page 274.

8.1.2 Writing XQuery Transformations

To create an XQuery transformation, implement an XQuery library module in the following namespace:

```
http://marklogic.com/rest-api/transform/yourTransformName
```

Where *yourTransformName* is the name used to refer to the transform in subsequent REST requests, such as installing and applying the transform.

The module must implement a public function called `transform` with the following interface:

```
declare function yourNS:transform(
  $context as map:map,
  $params as map:map,
  $content as document-node())
as document-node()
```

The table below describes the parameters:

Parameter	Description
$context	Service request context information such as input document type and URI, and output types accepted by the caller. For details, see “Context Map Keys” on page 276.
$params	Transformation specific parameters. The map contains a key for each transform parameter passed to the request for which the transform is applied. Define parameters when installing the transformation. For details, see “Installing Transformations” on page 277.
$content	The input document to which to apply the transformation. For example, the document being inserted into the database if the transform is applied during ingestion, or a `search:response` if the transform is applied during a search. For details, see “Expected Input and Output” on page 273.

The `map:map` type is a MarkLogic Server extended XQuery type representing a key-value map. For details, see Using the map Functions to Create Name-Value Maps in the *Application Developer's Guide*.

The expected output from your transform function depends upon the situation in which it is used. For details, see “Expected Input and Output” on page 273.

For a complete example, see “XQuery Example: Adding an Attribute During Ingestion” on page 280.

8.1.3 Writing XSLT Transformations

To create an XSLT transformation, implement an XSLT stylesheet that accepts the following parameters:

Parameter	Type	Description
`$context`	`map:map`	Service request context information such as input document type and URI, and output types accepted by the caller. For details, see "Context Map Keys" on page 276.
`$params`	`map:map`	Transformation specific parameters. The map contains a key for each transform parameter passed to the request for which the transform is applied. Define any parameters when installing the transformation. For details, see "Installing Transformations" on page 277.

The `map:map` type is a MarkLogic Server extended XQuery type representing a key-value map. For details, see Using the map Functions to Create Name-Value Maps in the *Application Developer's Guide*.

To use the `map` type in your stylesheet, include the namespace `http://marklogic.com/xdmp/map`. Reference the map contents using the `map:get` XQuery function.

The following example illustrates declaring the `map` namespace and `$context` and `$params` parameters, and referencing values in the `$params` map:

```
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:example="http://marklogic.com/rest-api/example/transform"
    xmlns:map="http://marklogic.com/xdmp/map">
<xsl:param name="context" as="map:map"/>
<xsl:param name="params"  as="map:map"/>
<xsl:template match="/*">
  <xsl:copy>
    <xsl:attribute name='{(map:get($params,"name"),"transformed")[1]}'
      select='(map:get($params,"value"),"UNDEFINED")[1]'/>
...
```

The expected output from your transform function depends upon the situation in which it is used. For details, see "Expected Input and Output" on page 273.

For a complete example, see "XSLT Example: Adding an Attribute During Ingestion" on page 282.

8.1.4 Expected Input and Output

The input document (`$content`) to a transform is always a document. The format and contents of the document depend on where the transform is used. The following table summarizes the expected input and output for a transform function for the supported use cases.

Transform Use Case	Input	Output
Document Ingestion	For JSON documents, the XML representation of the document, as described in Working With JSON in *Application Developer's Guide*. For all other document types, the input document, as received from the client application.	The document as it is to be stored in the database.
Document Retrieval	The document as stored in the database. For JSON documents, this means the XML representation of the document; for details see Working With JSON in *Application Developer's Guide*.	The document as it is to be returned to the client application. If the client requests a JSON document and the output from your transform is an XML document in the namespace `http://marklogic.com/xdmp/json/basic`, MarkLogic Server converts the XML to JSON when constructing the response.
Search	An XML document with a `<search:response/>` root element.	The document to put in the response body. If the client requested JSON results and the output is one of the following, MarkLogic Server converts the output from XML to JSON when constructing the response. • An XML document with a `<search:response/>` root element. • An XML document with a root element in the namespace `http://marklogic.com/xdmp/json/basic` that can be converted to JSON using json:transform-to-json.

Transform Use Case	Input	Output
Alert Match	An XML document with a `<rapi:rules/>` root element.	The document to put in the response body. If the client requested JSON results and the output is one of the following, MarkLogic Server converts the output from XML to JSON when constructing the response. • An XML document with a `<rapi:rules/>` root element. • An XML document with a root element in the namespace `http://mark-logic.com/xdmp/json/basic` that can be converted to JSON using json:transform-to-json.

8.1.5 Reporting Errors

This section discusses proper error reporting in your transform or service resource extension implementation. Use the approach described here for both XQuery and XSLT implementations. For more information about error reporting conventions, see "Error Reporting" on page 31.

To report an error from your implementation, use the `fn:error` XQuery function with the `RESTAPI-EXTNERR` error code, and provide the HTTP response code and additional error information in the 3rd parameter to `fn:error`.

The 3rd parameter to fn:error should be a sequence of the form `("status-code", "status-message", "payload-format", "response-payload")`. That is, when using `fn:error` to raise `REST-EXTNERR`, the `$data` parameter to `fn:error` is a sequence with the following members, all optional:

- HTTP status code. Default: 400.
- HTTP status message. Default: Bad Request.
- Payload Format. Allowed values: xml or json. If there is a payload format, there must be a payload.
- Response payload.

Note: Best practice is to use `RESTAPI-EXTNERR`. If you report any other error or raise any other exception, it is reported to the calling application as a 500 Server Internal Error.

For example, this resource extension function raises `RESTAPI-EXTNERR` if the input content type is not as expected:

```
declare function example:transform(
    $context as map:map,
    $params  as map:map,
    $input   as document-node()
) as document-node()
{
    (: get 'input-types' to use in content negotiation :)
    let $input-types := map:get($context,"input-types")
    let $negotiate :=
        if ($input-types = "application/xml")
        then () (: process, insert/update :)
        else fn:error((),"RESTAPI-EXTNERR",
          ("415","Raven","xml",xdmp:quote(<word>nevermore</word>)))
    return document { "Done"}  (: may return a document node :)
};
```

If a PUT request is made to the transform with an unexpected content type, the `fn:error` call causes the request to fail with a status 415 and to include the additional error description in the response body:

```
HTTP/1.1 415 Raven
Content-type: application/xml
Server: MarkLogic
Set-Cookie: SessionID=714070bdf4076536; path=/
Content-Length: 62
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<word>nevermore</word>
```

8.1.6 Context Map Keys

The context map parameter made available to XQuery and XSLT transformations can contain the following keys:

Key	Description
`input-type`	The MIME type of the input document.
`uri`	The URI of the input document.
`accept-types`	The MIME types accepted by the requestor. Multiple MIME types are separated by whitespace.
`output-type`	The expected output document MIME type. If the transform supports content negotiation, the transform function can set/modify this value.

8.1.7 Controlling Transaction Mode

This section discusses the default transaction mode under which your content transformation is evaluated, and how you can override the default behavior. This discussion assumes you are familiar with the MarkLogic Server transaction mode concept; for details, see Transaction Mode in *Application Developer's Guide*.

The default transaction mode for a transformation function depends on the request on whose behalf it is applied:

- PUT or POST to `/documents`: update
- GET `/documents`: query
- GET or POST for a query service (`/search`, `/qbe`, `/values`): query

Choosing the appropriate transaction mode can improve the resource utilization of your implementation. For example, an update transaction acquires exclusive locks on documents, as described in Update Transactions: Readers/Writers Locks in *Application Developer's Guide*. If you know that your transformation is actually a read-only operation, you can override the default transaction mode so that the function is evaluated in "query" transaction mode and does not acquire exclusive locks.

To override the default behavior, include a transaction mode annotation in the declaration of your function:

```
declare %rapi:transaction-mode("the-mode") function yourNS:transform(...)
```

The value for `%rapi:transaction-mode` can be either “update” or “query”.

For example, to specify query transaction mode for your `transform` function, use the following declaration:

```
declare %rapi:transaction-mode("query") function example:transform(
  $context as map:map,
  $params as map:map,
  $content as document-node())
as document-node()
{...};
```

8.2 Installing Transformations

This section covers the following topics:

- Basic Installation
- Defining Transform-Specific Parameters
- Including Transform Metadata

8.2.1 Basic Installation

To install or update a transformation, send a PUT request with a URL of the following form, with the transform code in the request body:

```
http://host:port/version/config/transforms/yourTransformName
```

Where *yourTransformName* is the name used when administering and applying the transformation.

Note: If you are installing an XQuery transform, *yourTransformName* must match the leaf name in the transform module namespace declaration. For example:

```
xquery version "1.0-ml";
module namespace example =
  "http://marklogic.com/rest-api/transform/yourTransformName";
...
```

Set the `Content-type` of the request to `application/xquery` for an XQuery transformation and `application/xslt+xml` for an XSLT transformation.

If your transform requires additional user-defined XQuery library modules, you must install them separately before installing your transform. For details, see “Using a Resource Service Extension” on page 293.

For a complete example, see “XSLT Example: Adding an Attribute During Ingestion” on page 282 or “XQuery Example: Adding an Attribute During Ingestion” on page 280.

8.2.2 Defining Transform-Specific Parameters

If the transform expects parameters, include the parameters as request parameters of the following form:

```
trans:paramName=paramTypeAndCardinality
```

Where *paramName* is the name of the parameter and *paramTypeAndCardinality* reflects the type and cardinality the parameter would have in an XQuery function declaration.

For example, "string?" means a string type parameter which may appear 0 or 1 time. The following URL illustrates a transform called "example" with an optional string parameter named "reviewer":

```
http://host:port/version/config/transforms/example?trans:reviewer=stri
ng\?
```

8.2.3 Including Transform Metadata

You can include metadata for `provider`, `version`, `title`, and `description` as request parameters. For example:

```
http://host:port/version/config/transforms/example?trans:reviewer=stri
ng\?&provider=MarkLogic%20Corporation
```

Metadata can be retrieved by sending a GET request to the /config/transforms service. See "Discovering Transformations" on page 279.

8.3 Applying Transformations

All MarkLogic REST API services that support transformations use the following mechanism for specifying application of a transformation:

1. Specify the name of an installed transform as the value of the `transform` request parameter:

   ```
   transform=transformName
   ```

2. If the transform expects parameters, specify each parameter with a request parameter of the form:

   ```
   trans:paramName=value
   ```

For example, to apply a transform during document insertion, you can send a PUT request to the `/documents` service with a `transform` request parameter. Assuming the transform called "example" with one parameter named "reviewer", the PUT request URL might look like the following:

```
http://host:port/version/documents?uri=/doc/theDoc.xml&transform=examp
le&trans:reviewer=me
```

You could apply the same transformation during document retrieval by sending a GET request to the `/documents` service with the same URL.

For a complete example, see "XSLT Example: Adding an Attribute During Ingestion" on page 282 or "XQuery Example: Adding an Attribute During Ingestion" on page 280.

8.4 Discovering Transformations

To discover the name, interface, and other metadata about installed transformations, send a GET request to the `/config/transforms` service of the form:

```
http://host:port/version/config/transforms
```

MarkLogic Server returns a summary of the installed transforms in XML or JSON. The default summary format is XML. Use the Accept header or `format` request parameter to select the output content type. For details, see "Controlling Input and Output Content Type" on page 25.

By default, this request rebuilds the transform metadata each time it is called to ensure the metadata is up to date. If you find this refresh makes discovery take too long, you can disable the refresh by setting the `refresh` request parameter to false. Disabling the refresh can result in this request returning inaccurate information, but it does not affect the "freshness" or availability of the implementation of any transforms.

The following example requests an XML report:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    -H "Accept: application/xml" \
    http://localhost:8003/v1/config/transforms
...
HTTP/1.1 200 Transform List Retrieved
Content-type: application/xml
Server: MarkLogic
Content-Length: 348
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<rapi:transforms xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:transform>
    <rapi:name>add-attr-xsl</rapi:name>
    <rapi:title/>
    <rapi:version/>
    <rapi:provider-name>MarkLogic Corporation</rapi:provider-name>
    <rapi:description/>
    <rapi:transform-parameters>
      <rapi:parameter>
        <rapi:parameter-name>value</rapi:parameter-name>
        <rapi:parameter-type>string?</rapi:parameter-type>
      </rapi:parameter>
      <rapi:parameter>
```

```
        <rapi:parameter-name>name</rapi:parameter-name>
        <rapi:parameter-type>string?</rapi:parameter-type>
      </rapi:parameter>
    </rapi:transform-parameters>
    <rapi:transform-source>
      /v1/transforms/add-attr
    </rapi:transform-source>
  </rapi:transform>
</rapi:transforms>
```

8.5 Retrieving the Implementation of a Transformation

To retrieve the XQuery code or XSLT stylesheet that implements a transformation, send a GET request to the `/config/transforms/{name}` service of the form:

```
http://host:port/version/config/transforms/transformName
```

Where *transformName* is the name under which the extension is installed.

Set the request Accept header to `application/xquery` for an XQuery transform. Set the request Accept header to `application/xslt+xml` for an XSLT transform.

MarkLogic Server returns the XQuery library module or XSLT stylesheet that implements the transform in the response body. For example:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
    -H "Accept: application/xquery" \
    http://localhost:8003/v1/config/transforms/example
...
xquery version "1.0-ml";
module namespace
example="http://marklogic.com/rest-api/transform/example";
declare function example:transform( ...
```

8.6 XQuery Example: Adding an Attribute During Ingestion

This example demonstrates using a transformation to add an attribute to the root node of an XML document during ingestion. The name and value of the added attribute are passed in as parameters to the transform function. You can use the same transform for document retrieval by applying it to a GET request instead of a PUT.

The transformation XQuery module shown below reads the attribute name and value from parameters, and then makes a copy of the input document, adding the requested attribute to the root node. The transformation cannot perform direct transformations on the input XML, such as calling `xdmp:node-replace`, because the document has not yet been inserted into the database when the transform function runs.

```
xquery version "1.0-ml";
module namespace example =
```

```
  "http://marklogic.com/rest-api/transform/add-attr";

declare function example:transform(
  $context as map:map,
  $params as map:map,
  $content as document-node()
) as document-node()
{
  if (fn:empty($content/*)) then $content
  else
    let $value := (map:get($params,"value"),"UNDEFINED")[1]
    let $name := (map:get($params, "name"), "transformed")[1]
    let $root  := $content/*
    return document {
      $root/preceding-sibling::node(),
      element {fn:name($root)} {
        attribute { fn:QName("", $name) } {$value},
        $root/@*,
        $root/node()
      },
      $root/following-sibling::node()
    }
};
```

To exercise this example:

1. Create a MarkLogic REST API instance on port 8003. For instructions, see “Creating an Instance” on page 33.

2. Cut and paste the code above into a file called “add-attr.xqy”.

3. Make a PUT request to the `/config/transforms/{name}` service to install the transform as “add-attr”, accepting 2 string parameters called “name” and “value”. For example:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT -d@"./add-attr.xqy" \
    -H "Content-type: application/xquery" \
    'http://localhost:8003/v1/config/transforms/add-attr?trans:name=st
ring\?&trans:value=string\?'
```

4. Optionally, verify installation of the transformation by making a GET request to the `/config/transforms` service. For example:

```
$ curl --anyauth --user user:password -X GET \
    -H "Accept: application/xml" \
    http://localhost:8003/v1/config/transforms
...
HTTP/1.1 200 Transform List Retrieved
Content-type: application/xml
Server: MarkLogic
Content-Length: 348
Connection: close
```

```
<?xml version="1.0" encoding="UTF-8"?>
<rapi:transforms xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:transform>
    <rapi:name>add-attr</rapi:name>
    <rapi:title/>
    <rapi:version/>
    <rapi:provider-name/>
    <rapi:description/>
    <rapi:transform-parameters>
      <rapi:parameter>
        <rapi:parameter-name>value</rapi:parameter-name>
        <rapi:parameter-type>string?</rapi:parameter-type>
      </rapi:parameter>
      <rapi:parameter>
        <rapi:parameter-name>name</rapi:parameter-name>
        <rapi:parameter-type>string?</rapi:parameter-type>
      </rapi:parameter>
    </rapi:transform-parameters>
    <rapi:transform-source>
      /v1/transforms/add-attr
    </rapi:transform-source>
  </rapi:transform>
</rapi:transforms>
```

5. Make a PUT request to the `/documents` service to insert a document into the database, using the "add-attr" transform to add the attribute "example" with a value of "new". For example:

```
$ curl --anyauth --user user:password -X PUT -d"<data/>" \
    'http://localhost:8003/v1/documents?uri=/doc/transformed.xml&trans
form=add-attr&trans:name=example&trans:value=new'
```

6. Make a GET request to the `/documents` service to retrieve the inserted document. Note the added "example" attribute. For example:

```
$ curl --anyauth --user user:password -X GET \
    http://localhost:8003/v1/documents?uri=/doc/transformed.xml
<?xml version="1.0" encoding="UTF-8"?>
<data example="new"/>
```

8.7 XSLT Example: Adding an Attribute During Ingestion

This example demonstrates using a transformation to add an attribute to the root node of an XML document during ingestion. The name and value of the added attribute are passed in as parameters to the stylesheet. You can use the same transform for document retrieval by applying it to a GET request instead of a PUT.

The transformation stylesheet shown below reads the attribute name and value from parameters, and then makes a copy of the input document, adding the requested attribute to the root node.

```
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:example="http://marklogic.com/rest-api/example/transform"
    xmlns:map="http://marklogic.com/xdmp/map">
<xsl:param name="context" as="map:map"/>
<xsl:param name="params"  as="map:map"/>
<xsl:template match="/*">
    <xsl:copy>
        <xsl:attribute
name='{(map:get($params,"name"),"transformed")[1]}'
            select='(map:get($params,"value"),"UNDEFINED")[1]'/>
        <xsl:copy-of select="@*"/>
        <xsl:copy-of select="node()"/>
    </xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

To exercise this example:

1. Create a MarkLogic REST API instance on port 8003. For instructions, see “Creating an Instance” on page 33.

2. Cut and paste the stylesheet above into a file called “add-attr.xsl”.

3. Make a PUT request to the `/config/transforms/{name}` service to install the transform as “add-attr-xsl”, accepting 2 string parameters called “name” and “value”. For example:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT -d@"./add-attr.xsl" \
    -H "Content-type: application/xslt+xml" \
    'http://localhost:8003/v1/config/transforms/add-attr-xsl?trans:nam
e=string\?&trans:value=string\?'
```

4. Optionally, verify installation of the transformation by making a GET request to the `/config/transforms` service. For example:

```
$ curl --anyauth --user user:password -X GET \
    -H "Accept: application/xml" \
    http://localhost:8003/v1/config/transforms
...
HTTP/1.1 200 Transform List Retrieved
Content-type: application/xml
Server: MarkLogic
Content-Length: 348
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<rapi:transforms xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:transform>
    <rapi:name>add-attr-xsl</rapi:name>
    <rapi:title/>
    <rapi:version/>
```

```
    <rapi:provider-name/>
    <rapi:description/>
    <rapi:transform-parameters>
      <rapi:parameter>
        <rapi:parameter-name>value</rapi:parameter-name>
        <rapi:parameter-type>string?</rapi:parameter-type>
      </rapi:parameter>
      <rapi:parameter>
        <rapi:parameter-name>name</rapi:parameter-name>
        <rapi:parameter-type>string?</rapi:parameter-type>
      </rapi:parameter>
    </rapi:transform-parameters>
    <rapi:transform-source>
      /v1/transforms/add-attr
    </rapi:transform-source>
  </rapi:transform>
</rapi:transforms>
```

5. Make a PUT request to the `/documents` service to insert a document into the database, using the "add-attr-xsl" transform to add the attribute "example" with a value of "new". For example:

```
$ curl --anyauth --user user:password -X PUT -d"<data/>" \
    'http://localhost:8003/v1/documents?uri=/doc/transformed.xml&trans
form=add-attr-xsl&trans:name=example&trans:value=new'
```

6. Make a GET request to the `/documents` service to retrieve the inserted document. Note the added "example" attribute. For example:

```
$ curl --anyauth --user user:password -X GET \
    http://localhost:8003/v1/documents?uri=/doc/transformed.xml
<?xml version="1.0" encoding="UTF-8"?>
<data example="new"/>
```

9.0 Extending the REST API

You can extend the MarkLogic REST API by installing a resource service extension. This chapter covers the following topics:

- Understanding Resource Service Extensions
- Creating a Resource Service Extension
- Installing a Resource Service Extension
- Using a Resource Service Extension
- Controlling Access to a Resource Service Extension
- Discovering Resource Service Extensions
- Retrieving the Implementation of a Resource Service Extension
- Example Resource Service Extension
- Managing Dependent Libraries and Other Assets

9.1 Understanding Resource Service Extensions

Use the resource service extension capability to make new REST service endpoints available through a MarkLogic REST API instance. To make a resource service extension available, follow these steps, which are discussed in detail in this chapter.

1. Create an XQuery library module that implements the extension.

2. Install the extension in a MarkLogic REST API instance to make it available to applications.

3. Access the extension service through a `/resources/{extensionName}` service endpoint.

9.2 Creating a Resource Service Extension

This section covers the following topics:

- Guidelines for Writing Resource Service Extensions
- The Resource Extension Interface
- Returning JSON Documents
- Reporting Errors
- Setting the Response Status Code
- Setting Additional Response Headers
- Context Map Keys
- Controlling Transaction Mode

9.2.1 Guidelines for Writing Resource Service Extensions

Keep the following guidelines in mind when authoring a resource extension:

- Methods such as `put` and `post` that accept an input document receive an in-memory representation. MarkLogic Server builtin functions that operate on in-database documents, such as `xdmp:node-replace`, cannot be applied directly to the input document. To modify the structure of the input document, perform the modifications during construction of a copy of the input content.
- Your code executes in the transaction context of the HTTP request on whose behalf it is called. You should not commit or rollback the current transaction. You can use `xdmp:eval` and `xdmp:invoke` to perform operations in a different transaction.
- If your extension depends on other user-defined modules, those modules must be installed in the modules database of the MarkLogic REST API instance. For detail, see "Managing Dependent Libraries and Other Assets" on page 299.
- Report errors by throwing one of the exceptions provided by the MarkLogic REST API. For details, see "Reporting Errors" on page 288.

9.2.2 The Resource Extension Interface

To create a resource service extension, implement an XQuery library module based on the template below. The module namespace, function names, and function signatures must be exactly as they appear in the template, with the following exceptions:

- The *extensionName* in the module namespace is the user-defined name under which the extension is installed and by which applications access the service. Each extension should have a unique name within the MarkLogic REST API instance in which it is installed.
- *yourNSPrefix* is a namespace prefix of your choosing.
- You only need to implement functions for the HTTP verbs supported by the extension.

Base your extension implementation on the following template. You must use the function localnames shown (`get`, `put`, etc.).

```
xquery version "1.0-ml";
module namespace yourNSPrefix =
    "http://marklogic.com/rest-api/resource/extensionName";

declare function yourNSPrefix:get(
    $context as map:map,
    $params  as map:map
) as document-node()*

declare function yourNSPrefix:put(
    $context as map:map,
    $params  as map:map,
    $input   as document-node()*
) as document-node()?
```

```
declare function yourNSPrefix:post(
    $context as map:map,
    $params  as map:map,
    $input   as document-node()*
) as document-node()*

declare function yourNSPrefix:delete(
    $context as map:map,
    $params  as map:map
) as document-node()?
```

The `map:map` type is a MarkLogic Server extended XQuery type representing a key-value map. For details, see Using the map Functions to Create Name-Value Maps in the *Application Developer's Guide*.

Your methods can accept input arguments. You can define the argument names and types when installing your extension; for details see "Installing a Resource Service Extension" on page 291. Arguments are passed to your methods through the `$params` map. See the `get` implementation in "Example Resource Service Extension" on page 295.

The following template summarizes the meaning of the function parameters:

Parameter	Description
`$context`	Service request context information such as input document type and URI, and output types accepted by the caller. For details, see "Context Map Keys" on page 290.
`$params`	Extension-specific parameters. The map contains a key for each parameter on a request to the extension service. Define parameters when installing the extension. For details, see "Installing a Resource Service Extension" on page 291.
`$input`	The input document to which to apply the transformation.

Your extension can return XML, JSON, text, or binary documents. Set `output-types` in the `$context` map to the MIME type appropriate for the returned content. The content is returned as-is, with the MIME type you specify.

9.2.3 Returning JSON Documents

This section covers important information for extensions that support JSON documents. To return a constructed JSON document, create a text document containing JSON and set `output-types` to `application/json` in the `$context` map. For example:

```
declare function example:get(
    $context as map:map,
    $params  as map:map
) as document-node()* {
    map:put($context,"output-types","application/json"),
    document { text { '{"key":"value"}' } }
};
```

If your extension operates on a document previously inserted into the database as a JSON document using the `/documents` service, the document will be in the internal XML representation when you access it in your extension function. If you want to return such a document from your extension as JSON, you are responsible for converting it back to JSON using the XQuery builtin `json:transform-to-json`. For example:

```
declare function example:get(
    $context as map:map,
    $params  as map:map
) as document-node()* {
    map:put($context,"output-types","application/json"),
    document { text {
      json:transform-to-json(fn:doc("/example/from-json.xml"))
    } }
};
```

For details, see Working With JSON in *Application Developer's Guide*.

9.2.4 Reporting Errors

Extensions and transforms use the same mechanism to report errors to the calling application: Use `fn:error` to report a `RESTAPI-EXTNERR` error, and provide additional information in the `$data` parameter of `fn:error`. You can control the response status code, status message, and provide an additional error reporting response payload. For example:

```
fn:error((),"RESTAPI-EXTNERR",
  (415, "Unsupported Input Type", "xml",
   xdmp:quote(<message>Only application/xml is supported</message>)))
```

For details, see "Reporting Errors" on page 274.

9.2.5 Setting the Response Status Code

By default, your resource service extension does not need to specify the HTTP response status code. If your implementation does not report an error, MarkLogic Server returns a `200 OK` response status.

To return a different status code, set the numeric code and message in `$context`. Use the key `output-status` with a `(code, message)` pair as its value.

> **Note:** Do not use `output-status` to report an error to the client. Instead, use `fn:error`. For details, see "Reporting Errors" on page 288.

The following example returns a `201 Created` status to the client instead of `200 OK`:

```
declare function example:put(
    $context as map:map,
    $params  as map:map,
    $input   as document-node()*
) as document-node()?
{
    ...
    map:put($context, "output-status", (201, "Created"));
};
```

9.2.6 Setting Additional Response Headers

The REST API sets headers such as Content-type and Content-Length for you when constructing the response from a resource extension request. You can specify additional headers by setting `output-headers` in the `$context` map.

> **Note:** Do not use `output-headers` to set Content-type. Use `output-types` to specify MIME type(s) instead.

The following example adds headers named "X-My-Header1" and "X-My-Header2" to the response by setting the output-headers to a sequence of the form (*header-name*, *header-value*, *header-name*, *header-value*, ...)

```
declare function example:put(
    $context as map:map,
    $params  as map:map,
    $input   as document-node()*
) as document-node()?
{
    ...
    map:put($context, "output-headers",
      ("X-MyHeader1", "header1 value", "X-MyHeader2", "header2 value"))
2"));
};
```

You can also set `output-headers` using a map with the header names as keys. The following code snippet has the same effect on the response headers as the previous example.

```
let $headers := map:map()
...
(map:put($headers, "X-MyHeader1", "header1 value"),
```

```
 map:put($headers, "X-MyHeader2", "header2 value"),
 map:put($context, "output-headers", $headers))
```

In both cases, the resulting response headers are similar to the following:

```
HTTP/1.1 200 OK
X-MyHeader1: header1 value
X-MyHeader2: header2 value
Server: MarkLogic
Content-Type: text/plain; charset=UTF-8
Content-Length: 4
Connection: Keep-Alive
Keep-Alive: timeout=5
```

9.2.7 Context Map Keys

The `$context` parameter to resource service extension functions can contain the following keys. Not all keys are always present. For example, there is no input-type on a GET request.

Key	Description
`input-types`	The MIME type of the input document(s). If there are multiple input documents, the value is a sequence of strings, one for the MIME type of each input document.
`accept-types`	For a GET, the MIME types accepted by the requestor. If there are multiple MIME types, the value is a sequence of strings.
`output-headers`	Response headers to include extra headers in the response to the client. Optional. Specify headers as a sequence of alternating header and value pairs: `(header1, value1, header2, value2, ...)`, or as a `map`. For details, see “Setting Additional Response Headers” on page 289.
`output-status`	The HTTP response status code and message to return to the client. Optional. If unset, 200 OK is returned, unless an error is raised. For example, to return a 201 Created status, set `output-status` to `(201, "Created")`. For details, see “Setting the Response Status Code” on page 288.
`output-types`	The output document MIME type. The service must set `output-types` for all output documents.

For a complete example of a resource service extension, see “Example Resource Service Extension” on page 295.

9.2.8 Controlling Transaction Mode

This section discusses the default transaction mode under which the implementation of a resource service extension is evaluated, and how you can override the default behavior. This discussion assumes you are familiar with the MarkLogic Server transaction mode concept; for details, see Transaction Mode in *Application Developer's Guide*.

Choosing the appropriate transaction mode can improve the resource utilization of your implementation. For example, the function implementing your POST method runs in "update" transaction mode by default. This means it acquires exclusive locks on documents, as described in Update Transactions: Readers/Writers Locks in *Application Developer's Guide*. If you know that your POST implementation is actually a read-only operation, you can override the default transaction mode so that the function is evaluated in "query" transaction mode and does not acquire exclusive locks.

The XQuery functions in a resource service extension implementation are evaluated using the following default transaction modes:

- `get` - query
- `put` - update
- `post` - update
- `delete` - update

To override the default behavior, include a transaction mode annotation in the declaration of your function:

```
declare %rapi:transaction-mode("the-mode") function yourNS:method(...)
```

The value for `%rapi:transaction-mode` can be either "update" or "query".

For example, to specify query transaction mode for your `put` function, use the following declaration:

```
declare %rapi:transaction-mode("query") function example:put(
    $context as map:map,
    $params  as map:map,
    $input   as document-node()*
) as document-node()?
{...};
```

9.3 Installing a Resource Service Extension

You install a resource service extension XQuery module using the MarkLogic REST API. Any dependent user-defined modules must be installed separately before installing your extension; for details, see "Managing Dependent Libraries and Other Assets" on page 299.

To install or update an extension, send a PUT request to `/config/resources` with a URL of the following form, with the extension code in the request body:

```
http://host:port/version/config/resources/extensionName
```

Where *extensionName* is the name used when administering and accessing the extension.

Note: *extensionName* must match the name in the extension module namespace declaration. For example:

```
xquery version "1.0-ml";
module namespace yourNSPrefix =
  "http://marklogic.com/rest-api/resource/extensionName";
...
```

Set the Content-type of the request body to `application/xquery`.

MarkLogic Server parses the extension implementation to determine which HTTP methods it supports. Only functions in the implementation that conform to the expected interface become available for use. For interface details, see “The Resource Extension Interface” on page 286.

If the extension service expects parameters, you can optionally “declare” the parameters using request parameters when installing the extension. This information is metadata that can be returned by a GET request to `/config/resources`. It is not used to check parameters on requests to the extension.

To include parameter metadata, declare each supported method and parameter using request parameters of the following form:

```
method=methodName

methodName:paramName=paramTypeAndCardinality
```

Where *methodName* is one of the extension functions: `get`, `put`, `post`, or `delete`; *paramName* is the name of the parameter; and *paramTypeAndCardinality* reflects the type and cardinality the parameter would have in an XQuery function declaration.

For example, “`xs:string?`” means a string type parameter which may appear 0 or 1 time. The following URL illustrates an extension named “example” with an optional string parameter named “reviewer” to the `put` method and a required string parameter “genre” to the `get` method:

```
'http://host:port/version/config/resources/example?method=put&put:revi
ewer=string\?&method=get&get:genre=xs:string'
```

You can also specify extension metadata for provider, version, title, and description as request parameters during installation. For example:

```
http://host:port/version/config/resources/example?method=get&get:genre
=string&provider=MarkLogic%20Corporation
```

For a complete example, see “Example Resource Service Extension” on page 295.

Note: If your module depends upon other user-defined modules, you must install them manually in the modules database of the REST API instance or in the Modules directory of your MarkLogic Server installation. For details, see “Using a Resource Service Extension” on page 293.

9.4 Using a Resource Service Extension

To access extension services, send a request of the appropriate type (GET, PUT, POST, or DELETE) to a URL of the following form:

```
http://host:port/version/resources/extensionName
```

Where *extensionName* is the name under which the extension is installed.

If the service expects parameters, supply the parameter names and values as request parameters, prefixing each parameter name with `rs:`. For example, if the “example” service for GET expects a parameter named “genre”, the request URL would be similar to the following:

```
http://localhost:8003/v1/resources/example?rs:genre=literature
```

The names, types, and cardinality of method parameters are resource specific. You can send a GET request to `/config/resources` to dynamically discover resources extension interfaces. For details, see “Discovering Resource Service Extensions” on page 293.

9.5 Controlling Access to a Resource Service Extension

Only users with the `rest-extension-user` role (or equivalent privileges) can execute modules in the modules database, including installed REST resource service extensions.

The pre-defined `rest-reader`, `rest-writer`, and `rest-admin` roles inherit `rest-extension-user`. You do not need additional configuration to grant access to your extension to users with these roles.

However, if instead of using the pre-defined roles, you create a custom role with the equivalent privileges, then you must explicitly inherit the `rest-extension-user` role in your custom role. Users with the custom role cannot execute resource service extensions unless you do so.

For details, see “Controlling Access to Documents Created with the REST API” on page 57.

9.6 Discovering Resource Service Extensions

To discover the name, interface, and other metadata about installed resource extensions, send a GET request to the `/config/resources` service of the form:

```
http://host:port/version/config/resources
```

MarkLogic Server returns a summary of the installed extensions in XML or JSON. The default summary content type is XML. Use the Accept header or `format` request parameter to select the output content type. For details, see "Controlling Input and Output Content Type" on page 25.

The amount of information available about a given extension depends on the amount of metadata provided during installation of the extension. The name and methods are always available. Details such as provider, version, and method parameter information are optional.

By default, this request rebuilds the extension metadata each time it is called to ensure the metadata is up to date. If you find this refresh makes discovery take too long, you can disable the refresh by setting the `refresh` request parameter to false. Disabling the refresh can result in this request returning inaccurate information, but it does not affect the "freshness" or availability of the implementation of any extensions.

The following example requests an XML report. One extension is installed, called "example". The extension implements only a get method, which accepts a string parameter named "genre". The extension provider is "Acme Widgets".

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -H "Accept: application/xml" \
    -X GET http://localhost:8004/v1/config/resources
...
<rapi:resources xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:resource>
    <rapi:name>example</rapi:name>
    <rapi:title/>
    <rapi:version/>
    <rapi:provider-name>Acme Widgets</rapi:provider-name>
    <rapi:description/>
    <rapi:methods>
      <rapi:method>
        <rapi:method-name>get</rapi:method-name>
        <rapi:parameter>
          <rapi:parameter-name>genre</rapi:parameter-name>
          <rapi:parameter-type>string</rapi:parameter-type>
        </rapi:parameter>
      </rapi:method>
    </rapi:methods>
    <rapi:resource-source>/v1/resources/example</rapi:resource-source>
  </rapi:resource>
</rapi:resources>
```

9.7 Retrieving the Implementation of a Resource Service Extension

To retrieve the XQuery code that implements an extension, send a GET request to the `/config/resources/{name}` service of the form:

```
http://host:port/version/config/resources/extensionName
```

Where *extensionName* is the name under which the extension is installed.

Set the request Accept header to `application/xquery`.

MarkLogic Server returns the XQuery library module that implements the extension in the response body. For example:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -H "Accept: application/xquery" \
    -X GET http://localhost:8004/v1/config/resources/example
...
xquery version "1.0-ml";
module namespace
example="http://marklogic.com/rest-api/resource/example";
declare function example:get( ...
```

9.8 Example Resource Service Extension

This section demonstrates the implementation, installation and invocation of a resource service extension.

- Overview
- Installing the Example Extension
- Using the Example Extension
- Example Extension Implementation

9.8.1 Overview

This example implements all the HTTP methods supported by the resource extension interface. It demonstrates the use of input arguments, as well as setting content type and response status codes. The methods implemented by this extension do the following:

Method	Description

GET	Accepts one or more string values through an “arg1” parameter. Returns an XML document that contains the input argument values using the template:
	`<args>`
	`  <arg1>value</arg1>`
	`<args>`

POST	Logs a message.
PUT	Accepts XML as input and returns “Done”.
DELETE	Logs a message.

If your implementation function returns content, MarkLogic Server responds with a status 200. If your function raises no errors but does not return content, MarkLogic Server responds with status 204. If your function raises an error, the response code depends on the error; for details, see “Reporting Errors” on page 288.

9.8.2 Installing the Example Extension

Installing the example requires identifying all the implemented methods, as well as the argument expected by the get method. To install the example:

1. If you do not already have a REST API instance, create one. This example assumes an instance is running on localhost:8004.

2. Copy the code from “Example Extension Implementation” on page 298 into a file. You can use any file name. This example assumes `example.xqy`.

3. Install the example using a command similar to the following. The extension metadata such as title, version, provider, description, and method interface details are all optional.

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT \
    -H "Content-type: application/xquery" -d@"./example.xqy" \
    "http://localhost:8004/v1/config/resources/example"
```

Optionally, you can include metadata during installation so that a GET request to `/config/resources` returns more detail. For example:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT \
    -H "Content-type: application/xquery" -d@"./example.xqy" \
    "http://localhost:8004/v1/config/resources/example?version=1.0&pro
vider=marklogic&description=A simple resource
example&method=get&method=post&method=put&method=delete&get:arg1=xs:st
ring*"
```

The table below breaks down the request parameters so it is easier to see what metadata is included.

Request Parameter	Description
`version=1.0`	The extension version number.
`provider=marklogic`	The extension provider.
`description=A simple resource example`	A brief description of the extension.
`method=get`	The extensions supports GET requests.
`method=post`	The extension supports POST requests.
`method=put`	The extension supports PUT requests
`method=delete`	The extension supports DELETE requests
`get:arg1=xs:string*`	The GET method accepts an argument named “arg1” which can occur 1 or more times.

9.8.3 Using the Example Extension

The following example command invokes the GET method of the example extension, passing the string “super fantastic” as arguments to the GET implementation:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET \
  -H "Accept: application/xml" \
  "http://localhost:8004/v1/resources/example?rs:arg1=super fantastic"

<?xml version="1.0" encoding="UTF-8"?>
<args>
  <arg1>super fantastic</arg1>
</args>
```

The following example invokes the PUT method:

```
$ curl --anyauth --user user:password -X PUT -d "<input/>" \
    -H "Content-type: application/xml" \
    http://localhost:8004/v1/resources/example

Done
```

The POST and DELETE methods simply log a message when invoked. If you call them, check the MarkLogic Server error log for output.

9.8.4 Example Extension Implementation

```
xquery version "1.0-ml";

(: Namespace pattern must be:
 : "http://marklogic.com/rest-api/resource/{$rname}"
 : and prefix must match resource name :)
module namespace example =
  "http://marklogic.com/rest-api/resource/example";

declare default function namespace
  "http://www.w3.org/2005/xpath-functions";
declare option xdmp:mapping "false";

(: Conventions:
 : Module prefix must match resource name,
 : and function signatures must conform to examples below.
 : The $context map carries state between the extension
 : framework and the extension.
 : The $params map contains parameters set by the caller,
 : for access by the extension.
 :)

(: Function responding to GET method - must use local name 'get':)
declare function example:get(
    $context as map:map,
    $params  as map:map
) as document-node()*
{
    (: set 'output-types', used to generate content-type header :)
    let $output-types :=
        map:put($context,"output-types","application/xml")
    let $arg1 := map:get($params,"arg1")
    let $content :=
        <args>
            {for $arg in $arg1
             return <arg1>{$arg1}</arg1>
            }
        </args>
    return document { $content }
    (: must return document node(s) :)
};

(: Function responding to PUT method - must use local name 'put'. :)
declare function example:put(
    $context as map:map,
    $params  as map:map,
    $input   as document-node()*
) as document-node()?
{
    (: get 'input-types' to use in content negotiation :)
    let $input-types := map:get($context,"input-types")
    let $negotiate :=
```

```
        if ($input-types = "application/xml")
        then () (: process, insert/update :)
        else error((),"ACK",
          "Invalid type, accepts application/xml only")
    return document { "Done"}  (: may return a document node :)
};

(: Function responding to POST method - must use local name 'post'. :)
declare function example:post(
    $context as map:map,
    $params  as map:map,
    $input   as document-node()*
) as document-node()*
{
    xdmp:log("post!")
};

(: Func responding to DELETE method - must use local name 'delete'. :)
declare function example:delete(
    $context as map:map,
    $params  as map:map
) as document-node()?
{
    xdmp:log("delete!")
};
```

9.9 Managing Dependent Libraries and Other Assets

Use the `/ext` service to install, update, retrieve, and delete assets required by your REST application, including additional XQuery library modules required by resource extensions and transforms, and partial update replace library modules.

> **Note:** You can only use the `/ext` service with REST API instances created with MarkLogic Server 7 or later. If you REST API instance was created with an earlier version, refer to the documentation for that version.

The following topics are covered:

- Maintenance of Dependent Libraries and Other Assets
- Installing or Updating an Asset
- Referencing an Asset
- Removing an Asset
- Retrieving an Asset List
- Retrieving an Asset

9.9.1 Maintenance of Dependent Libraries and Other Assets

When you install or update a dependent library module or other asset as described in this section, the asset is replicated across your cluster automatically. There can be a delay of up to one minute between updating and availability.

MarkLogic Server does not automatically remove dependent assets when you delete the related extension or transform.

Since dependent assets are installed in the modules database, they are removed when you remove the REST API instance if you include the modules database in the instance teardown. For details, see “Removing an Instance” on page 37.

If you installed assets in a REST API instance using MarkLogic 6, they cannot be managed using the `/ext` service unless you re-install them using `/ext`. Reinstalling the assets may require additional changes because the asset URIs will change. If you choose not to migrate such assets, continue to maintain them according to the documentation for MarkLogic 6.

9.9.2 Installing or Updating an Asset

Use the `/ext` service to install any assets required by resource service extensions or transformations, and to install partial update replace library modules. Using this service installs an asset into the modules database associated with the REST API instance. If the REST API instance is part of a cluster, the asset is automatically propagated throughout the cluster.

To install or update an asset, send a PUT request to `/ext/{`*`directories`*`}/{`*`asset`*`}` with a URL of the following form:

```
http://host:port/version/ext/directories/asset?perm:role=capability
```

Where *`directories`* is one or more database directories and *`asset`* is the asset document name. Use the `perm` request parameter to specify additional permissions on the asset, beyond the permissions granted to rest-admin privileges that are always present. The asset is installed into the modules database with the URI `/ext/`*`directories/asset`*.

The following example installs the XQuery library module `my-lib.xqy` into the modules database with the URI `/ext/my/domain/my-lib.xqy`. Execute privileges are granted to the role `my-users`.

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X PUT -i -d @./my-lib.xqy \
  -H "Content-type: application/xquery" \
  "http://localhost:8004/v1/ext/my/domain/my-lib.xqy?perm:my-users:exe
cute"
```

You can insert assets into the modules database as XML, text, or binary documents. MarkLogic Server determines the document format by examining the `format` request parameter, the MIME type in the `Content-type` HTTP header, or the document URI extension, in that order. For example, if the `format` parameter is present, it determines the document type. If there is no `format` parameter and no `Content-type` header, the URI extension (.xml, .jpg, etc.) determines the document type, based on the MIME type mappings defined for your installation; see the Mimetypes section of the Admin Interface.

9.9.3 Referencing an Asset

To use a dependent library installed with `/ext` in your extension or transform module, use the same URI under which you installed the dependent library. For example, if a dependent library is installed with the URI `/ext/my/domain/my-lib.xqy`, then the extension module using this library should include an import of the following form:

```
import module namespace dep="mylib" at "/ext/my/domain/lib/my-lib.xqy";
```

9.9.4 Removing an Asset

You can remove either a single asset or all the assets in a directory in the modules database associated with your REST API instance.

To remove a single asset, send a DELETE request with a URL of the following form:

```
http://host:port/version/ext/directories/asset
```

Where *`directories`* is one or more database directories and *`asset`* is the asset document name. For example, the following command removes an XQuery library module with the URI `/ext/my/domain/my-lib.xqy`:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X DELETE -i\
  "http://localhost:8004/v1/ext/my/domain/my-lib.xqy
```

To remove all assets in a modules database directory, send a DELETE request with a URL of the following form:

```
http://host:port/version/ext/directories/
```

Note: The URL must end with a terminating path separator (“/”) so that MarkLogic Server recognizes it as a directory.

Where *`directories`* is one or more database directories. For example, the following command removes all assets in the directory `/ext/my/domain/`:

```
$ curl --anyauth --user user:password -X DELETE -i\
  "http://localhost:8004/v1/ext/my/domain/
```

9.9.5 Retrieving an Asset List

You can retrieve a list of all assets in the modules database by sending a GET request with a URL of the following form:

```
http://host:port/version/ext/
```

You can retrieve a list of assets in a particular directory by sending a GET request with a URL of the following form, where *directories* is one or more database directory path steps:

```
http://host:port/version/ext/directories
```

You can request results as either XML or JSON, using either the HTTP Accept header or the `format` request parameter.

The following example retrieves a list of all assets installed under the database directory /ext/my/domain/, as XML:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET -i\
  -H "Accept: application/xml" \
  http://localhost:8004/v1/ext/my/domain/
...
HTTP/1.1 200 OK
Server: MarkLogic
Content-Type: text/xml; charset=UTF-8
Content-Length: 125
Connection: Keep-Alive
Keep-Alive: timeout=5

<rapi:assets xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:asset>/ext/patch/replace/my-lib.xqy</rapi:asset>
</rapi:assets>
```

The following example makes the same request, but returns JSON results.

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET -i\
  -H "Accept: application/json" \
  http://localhost:8004/v1/ext/my/domain/
...
HTTP/1.1 200 OK
Server: MarkLogic
Content-Type: text/plain; charset=UTF-8
Content-Length: 54
Connection: Keep-Alive
Keep-Alive: timeout=5

{"assets":[
  {"asset":"/ext/my/domain/my-lib.xqy"}
]}
```

9.9.6 Retrieving an Asset

To retrieve the contents of an asset document, send a GET request with a URL of the following form:

```
http://host:port/version/ext/directories/asset
```

Where *directories* is one or more database directories and *asset* is the asset document name. For example, the following command retrieves an XQuery library module with the URI `/ext/my/domain/my-lib.xqy`:

```
# Windows users, see Modifying the Example Commands for Windows
$ curl --anyauth --user user:password -X GET -i \
  -Accept: "application/xquery" \
  "http://localhost:8004/v1/ext/my/domain/my-lib.xqy
...
HTTP/1.1 200 Document Retrieved
Content-type: application/xquery
Server: MarkLogic
Content-Length: 633
Connection: Keep-Alive
Keep-Alive: timeout=5
...asset contents...
```

The response data content type is determined by the MIME type in the Accept header. No content conversions are applied. For example, if you retrieve a binary document containing PDF and set the Accept header to `application/xml`, the response Content-type will be `application/xml`, even though the data in the body is binary.

10.0 Migrating REST Applications

This chapter describes how to migrate an application hosted on a MarkLogic REST API instance from one MarkLogic Server installation to another, such as migrating an application from your development environment to a test or production environment.

> **Note:** These instructions only apply to applications that depend on a REST API instance, such as those created with Application Builder, the MarkLogic REST API, or the MarkLogic Java API.

The following topics are covered:

- Before You Begin
- Migrating the REST API Instance and Database Configuration
- Migrating the Contents of the Database

10.1 Before You Begin

Migrating a REST API application requires migrating the REST API instance (an App Server and modules database) and your content database. The REST API instance and the configuration of your content database are migrated using Configuration Manager. You can migrate the contents of your content database using any tool, but this guide assumes MarkLogic Content Pump (`mlcp`).

The migration procedure outlined by this chapter assumes you have the following:

- MarkLogic 7 on both the source and destination hosts.
- MarkLogic Content Pump (`mlcp`), XQSync, or an equivalent tool capable of copying content and metadata from one MarkLogic Server database to another. These instructions assume `mlcp`.

  To download and install mlcp, see Using MarkLogic Content Pump in the *Loading Content Into MarkLogic Server Guide*.

- `curl` or an equivalent tool for sending HTTP requests; see “Introduction to the curl Tool” on page 24. These instructions assume `curl`.
- Administrative user privileges on the source and destination MarkLogic Server instances.

10.2 Migrating the REST API Instance and Database Configuration

Use Configuration Manager to migrate your REST API instance and the configuration of your content database from to the destination MarkLogic Server cluster. If you are not familiar with Configuration Manager, see Using the Configuration Manager in *Administrator's Guide*.

- Export the Configuration from the Source Cluster
- Import the Configuration to the Destination Cluster

10.2.1 Export the Configuration from the Source Cluster

This procedure saves the configuration of your content database and REST API instance App Server, and the configuration and contents of the modules database to a ZIP file using Configuration Manager. For detailed instructions on using Configuration Manager, see Exporting a Configuration in *Administrator's Guide*.

> **Note:** You are not required to include the content database in this step, but if you do not, you must ensure it exists on the destination cluster before importing the REST API instance configuration.

1. Open the Configuration Manager on the source cluster in your browser. Authenticate as a user with the `manage-admin` role or equivalent privileges. For example, navigate to:

   ```
   http://source-host:8002/nav
   ```

2. Click on the Export tab. The Package Export interface is displayed.

3. If you want to migrate your content database configuration, place a check in the Settings column for your content database.

4. Click on Servers in the left column. Configuration Manager displays the App Servers available for export.

5. Place a check in the Settings and Modules columns for your REST API instance App Server.

6. Click the Export button in the upper right corner and select a location and name for the exported configuration ZIP file.

10.2.2 Import the Configuration to the Destination Cluster

This procedure uses Configuration Manager and the ZIP file created in "Export the Configuration from the Source Cluster" on page 305 to re-create your content database and REST API instance on the destination MarkLogic Server cluster. For detailed instructions on using Configuration Manager, see Importing a Configuration in *Administrator's Guide*.

> **Note:** If you did not include your content database configuration in the ZIP file, you must ensure that the content database exists on the destination cluster before performing this procedure.

1. Open the Configuration Manager on the destination cluster in your browser. Authenticate as a user with admin privileges. For example, navigate to:

   ```
   http://dest-host:8002/nav
   ```

2. Click on the Import tab. The Package Import interface is displayed.

3. Click the Browse button and browse to the ZIP containing your configuration.

4. Click Compare to import the package. The package contents are displayed. No configuration changes have been made at this time.

5. Click Apply. Configuration Manager re-creates the packaged configurations and the contents of the modules database.

6. On the destination cluster, create any MarkLogic Server users and roles required by your application. Use the same user and role name as on the source host.

Your REST API instance is fully functional at this point, including any resource extensions, persistent query options, or transformations stored in the modules database. If you imported your content database, then the database and its forests exist, but the database is still empty.

You can use the Admin Interface or Configuration Manager to confirm the existence of your REST API instance and content database. If you application includes resource extensions, transformations, or persistent query options, you can query them. For example, you can get a list of persistent query options using `GET /v1/config/query`.

10.3 Migrating the Contents of the Database

Use this procedure to migrate the contents of your application content database to the destination cluster. The database must already exist on the destination cluster using `mlcp`. You can also use XQSync or a similar tool to copy the contents.

1. Create an XDBC App Server attached to the content database in the source cluster. For details, see Creating a New XDBC Server in the *Administrator's Guide*.

2. Create an XDBC App Server attached to the content database in the destination cluster.

3. If the source and destination hosts are both reachable from the host where you perform the copy, use the mlcp `copy` command, as described in Copying Content Between Databases in the *Loading Content Into MarkLogic Server Guide*. For example:

```
$ mlcp.sh copy -mode local \
    -input_host src-host -input_port src-xdbc-port \
    -input_username user -input_password password \
    -output_host dest-host -output_port dest-xdbc-port \
    -output_username user -output_password password
```

4. If you cannot reach both the source and destination host from the host where you perform the copy, use the mlcp `export` and `import` commands, as described in Exporting to an Archive and Importing Content Into MarkLogic Server in the *Loading Content Into MarkLogic Server Guide*. For example:

```
# Export the source database content to an archive
$ mlcp.sh export -mode local -output_type archive \
```

```
    -host src-host -port src-xdbc-port \
    -username user -password password \
    -output_file_path an-existing-directory
# One or more ZIP files are created in the output directory.

# Copy the archive files to a host that can reach the destination host

# Import the archive to the destination database
$ mlcp.sh import -mode local -input_file_type archive \
    -host dest-host -port dest-xdbc-port \
    -username user -password password
    -input_file_path dir-containing-archive
```

You can now remove the XDBC App Servers created during this procedure.

11.0 Appendix A: JSON Query Options Reference

This chapter defines the JSON structure of query options (`search:options`) used by the search related features of the MarkLogic REST API and the MarkLogic Java API.

Query options can also be expressed as XML. For details, see "Appendix B: XML Query Options Reference" on page 366.

- Query Options
- attribute-descriptor
- bucket
- collection-constraint
- computed-bucket
- constraint
- container-constraint
- custom-constraint
- default-suggestion-source
- element-descriptor
- extract-metadata
- field-descriptor
- geo-attr-pair-constraint
- geo-elem-constraint
- geo-elem-pair-constraint
- geo-path-constraint
- grammar
- heatmap-descriptor
- joiner
- operator
- path-expression
- properties-constraint
- range-constraint
- result-decorator
- sort-order
- starter

- suggestion-source
- term-definition
- transform-results
- value-constraint
- values-or-tuples
- word-constraint
- word-lexicon
- facet-options
- term-options
- values-options
- geo-options
- range-option
- suggestion-option

11.1 How to Use This Reference

This reference describes the layout of the options structure usable with the query capabilities of the MarkLogic REST API and MarkLogic Java API, when expressed as JSON. The structures are shown with all possible properties, but not all properties may be usable together.

For specific semantics, refer to the description of each structure. For more details, refer to the options descriptions for the XQuery function `search:search` in the *XQuery and XSLT Reference Guide*.

11.2 Query Options

```
"options": {
  "additional-query": [ string ],
  "concurrency-level": string,
  "constraint": [ constraint ],
  "debug": boolean,
  "default-suggestion-source": default-suggestion-source,
  "extract-metadata": extract-metadata,
  "forest": unsigned long,
  "fragment-scope": string,
  "grammar": grammar,
  "operator": [ operator ],
  "page-length": unsigned long,
  "quality-weight": double,
  "result-decorator": result-decorator,
  "return-aggregates": boolean,
  "return-constraints": boolean,
  "return-facets": boolean,
```

```
    "return-frequencies": boolean,
    "return-metrics": boolean,
    "return-plan": boolean,
    "return-qtext": boolean
    "return-query": boolean,
    "return-results": boolean,
    "return-similar": boolean,
    "return-values": boolean,
    "search-option": [ string ],
    "searchable-expression": { path-expression }
    "sort-order": [ sort-order ],
    "suggestion-source": [ suggestion-source ],
    "term": term-definition,
    "transform-results": transform-results,
    "tuples": values-or-tuples,
    "values": values-or-tuples
  }
```

Key	Value	Description

additional-query[]	[*string*]	Additional serialized `cts:query`'s, as XML literals. The queries are AND'd with the string and/or structured query that is supplied to the request. The query results are constrained by the `additional-query`, but any terms matching the addtional-query are not highlighted in the snippet result output. For example, the following constrains the results to the directory named /my/directory/:
		`"additional-query":[`
		`  "<directory-query`
		`  xmlns='http://marklogic.com/cts'>`
		`  <uri>/my/dir/</uri>`
		`  </directory-query>"`
		`]`

concurrency-level	*unsigned integer*	The maximum number of threads used to resolve facets. The default is 8, which specifies that at most 8 threads will be used concurrently to resolve facets.
constraint[]	[constraint]	Constraints define facets which can be returned as part of the search results. The default is no defined constraints. Each constraint must have an name, which is the unique name of this constraint. The name can be used in the search grammar to specify the constraint. The list can contain zero or more constraint objects.
debug	*boolean*	Activates debugging mode. Additional report elements will be present in the output. Set to `true` to activate. The default is `false`.
default-suggestion-source	default-suggestion-source	Defines the content to be used as the default source of suggestions (see `search:suggest`).

Key	Value	Description
extract-metadata	extract-metadata	Specifies data to return in the `metadata` section of the `response` output. You can specify element or attribute content from the search hits to be returned or values from any constraints in your search.
forest[]	[*unsigned long*]	One or more forest IDs.
fragment-scope	*string*	Allowable values: • `properties` • `documents` (default) Controls the global fragment scope over which to search. The global scope applies both to what the search returns (that is, if it returns results from document fragments or from property fragments) and the global scope is inherited by any constraints that do not explicitly override the `fragment-scope`. You can override the global fragment scope by setting a local fragment scope inside a constraint or term definition. For example, a `fragment-scope` of `properties` on a range-constraint enables you to facet on a value stored in a property, even if you are searching over documents.
grammar	grammar	A custom search grammar definition. The default grammar defines "Google-style" parsing.
operator[]	[operator]	A list of state elements, each representing a unique run-time configuration option. For example, if an operator with the name "sort" is defined, query text [sort:foo] will select the `state` child with the name "foo" at query runtime, using the option specified on that state element. Options affecting query parsing (such as `constraint`, `grammar`, `term`, `empty`) may not be configured via operators.
page-length	*unsigned long*	Specifies the number of results per page. The default value is 10.

Key	Value	Description
quality-weight	*double*	Specifies a a weighting factor to use in the query. The default value is 1.0.
return-aggregates	*boolean*	Include the result of running a builtin or user-defined aggregate function. Applies only to queries against values or tuples.
return-constraints	*boolean*	Include original constraint definitions in the results. The default is false.
return-facets	*boolean*	Include resolved facets in the results. The default is true.
return-frequencies	*boolean*	Include frequencies in the results.
return-metrics	*boolean*	Include statistics in the results. The default is true.
return-plan	*boolean*	Include `xdmp:plan` output in the results. The default is false. The output is useful in examining the evaluation plan of the search.
return-qtext	*boolean*	Include the original query text in the results. The default is true.
return-query	*boolean*	Include the XML query representation in the results. The default is false.
return-results	*boolean*	Include search results in the output. (Use transform-results to specify how each result should be formatted.) The default is true.
return-similar	*boolean*	Include with each search result a list of URLs of similar documents in the database. The default is false.
return-values	*boolean*	When querying a range index or values lexicon, whether or not to include the index/lexicon values in the results. Default: true.

Key	Value	Description
search-option[]	[*string*]	For advanced users, one or more options to be passed in to the query operation (for example, `filtered`, `unfiltered`, `score-logtfidf`, and so on). See the XQuery function `cts:search`.
searchable-expression	path-expression	An expression to be searched. Whatever expression is specified is returned from the search. For example, if you specify `//p`, then `p` elements that match the search criteria are returned. The expression must be an inline fully searchable XPath expression, and all necessary namespaces must be declared. The default value is `fn:collection()`, which searches all documents in the database. This option does not affect facet results. To constrain facets, use the `additional-query` option.
sort-order[]	[sort-order]	Set the default sort order. The first such value is the primary sort order, the second secondary sort order, and so on. The default is to sort by score, descending. Note that the default scoring algorithm can be set just like any other option with the option named `search-option`.
suggestion-source	[suggestion-source]	Specifies a constraint source to override a named constraint when using `search:suggest`.
transform-results	transform-results	Specifies a function to use to process a search result for the snippet output. The default is that each result is formatted using the built-in default snippetting function.
tuples	values-or-tuples	Defines one or more value lexicons query against, matching value co-occurrences. That is, tuples of values, each of which appear in the same fragment.
values	values-or-tuples	Defines one or more value lexicons to query against.

11.3 attribute-descriptor

```
{
  "name": string,
  "ns": string
}
```

The definition of an XML element attribute. An attribute descriptor is generally accompanied by an element descriptor to identify the containing element.

Key	Value	Description
name	*string*	Required. The name of an attribute.
ns	*string*	The namespace in which the attribute is defined.

Example:

```
{ "name": "bar", "ns": "http://marklogic.com/examples" }
```

11.4 bucket

```
{
  "name": string,
  "label": string,
  "lt": atomic-value,
  "ge": atomic-value
}
```

A named range of static values within a range constraint. For dynamic value ranges see computed-bucket. Values assigned to a bucket meet the criteria:

```
bucket.lt < value <= bucket.gt
```

Key	Value	Description
name	*string*	Required. Identifies this bucket.
label	*string*	Text that can be used to label the bucket. Corresponds to the text() in an XML `<bucket/>` definition.
lt	*atomic-value*	The lower bound for values assigned to this bucket.
ge	*atomic-value*	The upper bound for values assigned to this bucket.

Example:

```
{
  "options": {
    "constraint": [
      {
        "name": "decade",
        "range": {
          "type": "xs:gYear",
          "facet": true,
          "facet-option": [
            "limit=10"
          ],
          "bucket": [
            {
              "lt": "1930",
              "ge": "1920",
              "name": "1920s",
              "label": "1920s"
            },
            {
              "lt": "1940",
              "ge": "1930",
              "name": "1930s",
              "label": "1930s"
            }
          ],
          "attribute": {
            "ns": "",
            "name": "year"
          },
          "element": {
            "ns": "http:\/\/marklogic.com\/wikipedia",
            "name": "nominee"
          }
        }
      }
```

```
    ]
  }
}
```

11.5 collection-constraint

```
{
  "prefix": string,
  "facet-option": [ string ],
  "facet": boolean
}
```

Key	Value	Description
prefix	*string*	Required. Constrain a query to documents in collections matching this prefix.
facet-option[]	[*string*]	Zero or more facet-options.
facet	*boolean*	Whether or not to generate facets from this constraint.

The following example defines a collection constraint called “my-collection” that constrains a query to documents in collections with the prefix “/my/path/”:

```
{
  "options": {
    "constraint": [
      {
        "name": "my-collection",
        "collection": {
          "prefix": "\/my\/path\/"
        }
      }
    ]
  }
}
```

11.6 computed-bucket

```
{
  "name": string,
  "label: string,
  "lt": atomic-value,
  "ge": atomic-value,
  "anchor": string,
  "lt-anchor": string,
```

```
  "ge-anchor": string
}
```

A named range of dynamic temporal values within a range constraint. For static value ranges see bucket. Values assigned to a bucket meet the criteria:

```
(bucket.anchor + bucket.lt) < value <= (bucket.anchor + bucket.gt)
```

Use `lt-anchor` and `ge-anchor` to override the anchor value used to compute the lower and upper bounds, respectively.

Key	Value	Description
name	*string*	Required. The name of this bucket.
label	*string*	Text that can be used to label the bucket. Corresponds to the text() in an XML `<bucket/>` definition.
lt	*atomic-value*	The lower bound for values assigned to this bucket, relative to `anchor` or `lt-anchor`. The value should be an `xs:duration`, `xs:yearMonthDuration`, or `xs:dayTimeDuration`.
ge	*atomic-value*	The lower bound for values assigned to this bucket, relative to `anchor` or `ge-anchor`. The value should be an `xs:duration`, `xs:yearMonthDuration`, or `xs:dayTimeDuration`.
anchor	*string*	A dynamic temporal anchor value for the bucket. The bucket bounds are relative to this dynamic value. Allowed values: • now • start-of-day • start-of-month • start-of-year
lt-anchor	*string*	Overrides `anchor` for computing the bucket lower bound. For allowed values, see `anchor`.
ge-anchor	*string*	Overrides `anchor` for computing the bucket upper bound. For allowed values, see `anchor`.

Example:

```
{
  "options": {
    "constraint": [
      {
        "name": "made",
        "range": {
```

```
            "type": "xs:dateTime",
            "computed-bucket": [
              {
                "name": "today",
                "ge": "P0D",
                "lt": "P1D",
                "anchor": "now",
                "label": "Today"
              },
              {
                "name": "30-days",
                "ge": "-P30D",
                "lt": "P1D",
                "anchor": "now",
                "label": "Last 30 days"
              },
              {
                "name": "60-days",
                "ge": "-P60D",
                "lt": "P1D",
                "anchor": "now",
                "label": "Last 60 Days"
              },
              {
                "name": "year",
                "ge": "-P1Y",
                "lt": "P1D",
                "anchor": "now",
                "label": "Last Year"
              }
            ],
            "element": {
              "ns": "http:\/\/example.com",
              "name": "manufactured"
            },
            "attribute": {
              "ns": "",
              "name": "date"
            }
          }
        }
      ]
    }
  }
```

11.7 constraint

```
{
  "name": string,
  "annotation": [ string ],
  "range": range-constraint,
  "word": word-constraint,
  "value": value-constraint,
```

```
    "collection": collection-constraint,
    "properties": properties-constraint,
    "container": container-constraint,
    "element-query": element-descriptor,
    "custom": custom-constraint,
    "geo-elem": geo-elem-constraint,
    "geo-elem-pair": geo-elem-pair-constraint,
    "geo-attr-pair": geo-attr-pair-constraint,
    "geo-path": geo-path-constraint
}
```

Every constraint must have a unique `name` and may have an annotation. The constraint semantics are expressed through one or more of the remaining constraint object types (range, word, value, etc.).

Key	Value	Description
name	*string*	Required. The name must be unique within these query options and may not contain whitespace.
annotation	[*string*]	Your comments. Annotations have no effect.
range	range-constraint	Specifies the element or attribute on which to constrain by range. There must be a range index of the specified type (and collation for string range indexes) defined.
word	word-constraint	Specifies the element, attribute, or field on which to constrain by word. You cannot create facets from a `word` constraint.
value	value-constraint	Specifies element or attribute values on which to constrain by value. You cannot create facets from a `value` constraint.
collection	collection-constraint	Specifies a collection which to constrain.
properties	properties-constraint	Specifies a constraint that finds matches on the corresponding properties documents. You cannot create facets from a `properties` constraint.
container	container-constraint	Specifies a constraint that restricts the search to the specified XML element or JSON object. You cannot create facets from a `container` constraint.

Key	Value	Description
element-query	element-descriptor	Specifies a constraint that restricts the search to the specified element. You cannot create facets from an `element-query` constraint.
custom	custom-constraint	Specifies a custom constraint along with the name of the function implementations used to evaluate the custom constraint.
geo-elem	geo-elem-constraint	Specifies a geospatial constraint. If the constraint has a `heatmap` child, then it can be used as a facet, and can contain `facet-options`.
geo-elem-pair	geo-elem-pair-constraint	Specifies a geospatial constraint. If the constraint has a `heatmap` child, then it can be used as a facet, and can contain `facet-options`.
geo-attr-pair	geo-attr-pair-constraint	Specifies a geospatial constraint. If the constraint has a `heatmap` child, then it can be used as a facet, and can contain `facet-options`.

11.8 container-constraint

```
{
  "element": element-descriptor,
  "json-key": string,
  "fragment-scope": string
}
```

Specifies an XML element or JSON object container constraint, identified by element QName or JSON key. The constraint must include exactly one of `element` or `json-key`.

Key	Value	Description
element	element-descriptor	Specifies the XML element container to which the search is constrained. If the constraint defines an `element`, it must not contain a `json-key`.

Key	Value	Description
json-key	*string*	Specifies the JSON key of the object to which the search is constrainted. If the constraint defines a `json-key`, it must not contain an `element`.
fragment-scope	*string*	Optional. Allowable values: • `properties` • `documents` (default) Set a local fragment scope for this constraint. The local `fragment-scope` overrides a global `fragment-scope`. For example, a `fragment-scope` of `properties` on a range constraint enables you to facet on a value stored in a property, even if you are searching over documents.

Example:

```
{"options": {
  "constraint": [{
    "name": "title",
    "container": {
      "element": {
        "ns": "",
        "name": "TITLE"
      }
    }
  }]
}}
```

11.9 custom-constraint

```
{
  "parse": {
    "apply": string,
    "ns": string,
    "at": string
  },
  "start-facet": {
    "apply": string,
    "ns": string,
    "at": string
  },
  "finish-facet": {
    "apply": string,
    "ns": string,
    "at": string
  },
  "facet-option": [ string ],
  "term-option": string,
```

```
    "facet": boolean
}
```

Specifies a custom constraint along with the name of the function implementations used to evaluate the custom constraint.

Key	Value	Description
parse	*object*	Required. Identifies the function used to parse the query text to produce a `cts:query`.
parse.apply	*string*	Required. The parsing function localname.
parse.ns	*string*	The namespace in which the function is declared.
parse.at	*string*	The URI of the XQuery library module implementing function.
start-facet	*object*	Identifies a function to make lexicon API calls that return the values and counts used to construct facets.
start-facet.apply	*string*	Required if `start-facet` is defined.
start-facet.ns	*string*	The namespace in which the function is declared.
start-facet.at	*string*	The URI of the XQuery library module implementing function.
finish-facet	*object*	Identifies the function takes input from the `start-facet` function (if it is used) and constructs the facet element.
finish-facet.apply	*string*	Required if `finish-facet` is defined. The function localname.
finish-facet.ns	*string*	The namespace in which the function is declared.
finish-facet.at	*string*	The URI of the XQuery library module implementing function.
facet-option[]	[*string*]	Zero or more facet-options.
term-option[]	[*string*]	Zero or more term-options.
facet	*boolean*	Whether or not to use this constraint for faceting.

11.10 default-suggestion-source

```
{
  "ref": string,
  "suggestion-option": [ suggestion-option ]
  "annotation": [ string ]
  "collection": collection-constraint,
  "range": range-constraint,
  "word": word-constraint,
  "word-lexicon": word-lexicon
}
```

Specifies a constraint source for generating suggestions for naked terms with `search:suggest`. Suggestions are often used for type-ahead suggestions in a search user interface. For terms qualified by a constraint prefix, such as “tag:value”, use the suggestion-source option. For more details, see default-suggestion-source Option in the *Search Developer’s Guide*.

You can either use `ref` to name a constraint defined elsewhere in the options, or omit `ref` and define a constraint within the `default-suggestion-source`. Use `suggestion-option` to override default behaviors such as sort order when defining a constraint within the `default-suggestion-source`.

> **Note:** The use of `word-lexicon` (the database-wide word lexicon) is not recommended as best practice; collection and range lexicons will yield the best performance.

The constraint defined in a `default-suggestion-source` can optionally include a `collation` value, which specifies the collation of the value lexicon used during query evaluation. If no collation is specified, then the query uses the default collation for the context in which the query is evaluated.

A `default-suggestion-source` can contain the following properties. The `collection`, `range`, `word`, and `word-lexicon` properties are mutually exclusive.

Key	Value	Description
`ref`	*string*	The name of a constraint defined in the same set of query options. Optional.
`suggestion-option`	[suggestion-option]	Query options to use in generating suggestions. Each option is a string. The supported options depend on the constraint type.
`annotation`	[*string*]	Your comments. Annotations have no effect.

Key	Value	Description
`range`	range-constraint	Defines a constraint that overrides the one named by `ref` when generating search suggestions.
`word`	word-constraint	Defines a constraint that overrides the one named by `ref` when generating search suggestions.
`collection`	collection-constraint	Defines a constraint that overrides the one named by `ref` when generating search suggestions.
`word-lexicon`	word-lexicon	Defines a constraint that overrides the one named by `ref` when generating search suggestions.

For example, the following options define the “color” constraint as the default suggestion source. The suggestion source for “color” specifies suggestions matches should be case-sensitive. When generating suggestions for an unqualified term such as “crim”, suggest are only drawn from the words that occur in the JSON property named “color”.

```
{ "options": {
    "constraint": [
      { "name": "color",
        "word": { "json-key": "color" }
      }
    ],
    "default suggestion-source": [
      { "ref": "color" }
    ]
} }
```

The following options define a range constraint as the default suggestion source and sets the sort order for the result suggestions to descending.

```
{ "options": {
    "default suggestion-source": [
      { "range": { "json-key": "species" },
        "suggestion-option": [ "descending" ]
      }
    ]
} }
```

11.11 element-descriptor

```
{
  "name": string,
```

```
  "ns": string
}
```

This object defines an XML element for use in options such as constraints.

Key	Value	Description
name	*string*	Required. The name of an element.
ns	*string*	The namespace in which the named element is defined.

Example:

```
{ "name": "foo", "ns": "http://marklogic.com/examples" }
```

11.12 extract-metadata

```
{
  "constraint-value": [
    { "ref": string }
  ],
  "qname": [
    {
      "elem-ns": string,
      "elem-name": string,
      "attr-ns": string,
      "attr-name": string
    }
  ]
  "json-key": [ string ]
}
```

Use `extract-metadata` to select one or more metadata elements from each document in the query result set. Selections are made by specifying qnames (element or element and attribute), JSON keys, or constraint references (collection, range and geospatial types only) for extraction from indexes.

Key	Value	Description
constraint-value[]	*array*	An array of references to named constraints defined in these query options. Corresponding indexes must exist when the query executes.
constraint-value.ref	*string*	The name of a collection, range, or geospatial constraint defined in these query options.
qname[]	*array*	An array of QName definitions for element or element attribute content from the search hits to be returned.
qname.elem-name	*string*	The name of an element from which content should be returned. Required (if using qname).
qname.elem-ns	*string*	The namespace URI in which `qname.elem-name` is defined. Required (if using qname)
qname.attr-name	*string*	The name of an attribute of `qname.elem-name` whose value should be returned.
qname.attr-ns	*string*	The namespace URI in which `qname.attr-name` is defined.
json-key[]	[*string*]	The names of json keys from which content should be returned. Applicable only to JSON documents loaded using the MarkLogic REST API or MarkLogic Java API.

Example:

```
{
  "options": {

    "extract-metadata": {
      "constraint-value": [
        { "ref": "title" },
        { "ref": "speaker" }
      ],
      "qname": [
        {
          "elem-ns": "http:\/\/example.com",
          "elem-name": "foo"
        }
      ],
      "json-key": [ "foo", "bar" ]
    }
```

```
    "constraint": [
      {
        "name": "title",
        "range": {
          "type": "xs:string",
          "element": {
            "ns": "",
            "name": "TITLE"
          }
        }
      },
      {
        "name": "speaker",
        "range": {
          "type": "xs:string",
          "element": {
            "ns": "",
            "name": "SPEAKER"
          }
        }
      }
    ]
  }
}
```

11.13 field-descriptor

```
{
  "name": string,
  "collation": string
}
```

Key	Value	Description
name	*string*	Required. The name of a field.
collation	*string*	An optional collation URI.

11.14 geo-attr-pair-constraint

```
{
  "parent": element-descriptor,
  "lat": attribute-descriptor,
  "lon": attribute-descriptor,
  "facet-option": [ string ],
  "geo-option": [ string ],
  "fragment-scope": string,
```

```
  "heatmap": heatmap-descriptor
}
```

A geospatial constraint modelling a geospatial index with coordinates stored as element attributes. By default coordinates are stored as latitude,longitude points. To reverse the coordinate order, add the geo-option "long-lat-points" to the query options configuration. For more details, see `cts:element-attribute-pair-geospatial-query`.

Key	Value	Description
parent	element-descriptor	Defines the parent element.
lat	*object*	The latitude-defining attribute.
lon	*object*	The longitude-defining attribute.
facet-option[]	[*string*]	Zero or more facet-options.
geo-option[]	[*string*]	Zero or more geo-options.
fragment-scope	*string*	Allowable values: • `properties` • `documents` (default) Set a local fragment scope for this constraint. The local `fragment-scope` overrides a global `fragment-scope`. For example, a `fragment-scope` of `properties` on a range constraint enables you to facet on a value stored in a property, even if you are searching over documents.
heatmap	heatmap-descriptor	A heatmap that to use for faceting.

Example:

```
{
  "options": {
    "constraint": [
      {
        "name": "my-geo-attr-pair",
        "geo-attr-pair": {
          "facet-option": [
            "empties"
          ],
          "heatmap": {
            "s": 23.2,
            "w": -118.3,
            "n": 23.3,
            "e": -118.2,
```

```
            "latdivs": 4,
            "londivs": 4
          },
          "parent": {
            "ns": "ns1",
            "name": "elem1"
          },
          "lat": {
            "ns": "ns2",
            "name": "attr2"
          },
          "lon": {
            "ns": "ns3",
            "name": "attr3"
          }
        }
      }
    ]
  }
}
```

11.15 geo-elem-constraint

```
{
  "parent": element-descriptor,
  "element": element-descriptor,
  "facet-option": [ string ],
  "geo-option": [ string ],
  "fragment-scope": string,
  "heatmap": heatmap-descriptor
}
```

A geospatial constraint modelling a geospatial index with coordinates stored in a single element. By default coordinates are stored as latitude,longitude points. To reverse the coordinate order, add the geo-option "long-lat-points" to the query options configuration. For more details, see `cts:element-geospatial-query`.

Key	Value	Description
parent	element-descriptor	The parent element.
element	element-descriptor	The element defining the latitude and longitude.

Key	Value	Description
facet-option[]	[*string*]	Zero or more facet-options.
geo-option[]	[*string*]	Zero or more geo-options.
fragment-scope	*string*	Allowable values: • `properties` • `documents` (default) Set a local fragment scope for this constraint. The local `fragment-scope` overrides a global `fragment-scope`. For example, a `fragment-scope` of `properties` on a range constraint enables you to facet on a value stored in a property, even if you are searching over documents.
heatmap	heatmap-descriptor	Describes a heatmap to use for faceting.

Example:

```
{
  "options": {
    "constraint": [
      {
        "name": "geo-elem-child",
        "geo-elem": {
          "parent": {
            "ns": "",
            "name": "g-elem-child-parent"
          },
          "element": {
            "ns": "",
            "name": "g-elem-child-point"
          }
        }
      }
    ]
  }
}
```

11.16 geo-elem-pair-constraint

```
{
  "parent": element-descriptor,
  "lat": element-descriptor,
  "lon": element-descriptor,
  "facet-option": [ string ],
  "geo-option": [ string ],
  "fragment-scope": string,
  "heatmap": heatmap-descriptor
}
```

A geospatial constraint modelling a geospatial index with coordinates stored in a two elements. By default coordinates are stored as latitude,longitude points. To reverse the coordinate order, add the geo-option "long-lat-points" to the query options configuration. For details, see `cts:element-pair-geospatial-query`.

Key	Value	Description
parent	element-descriptor	The parent element.
lat	element-descriptor	The latitude-defining element.
lon	element-descriptor	The longitude-defining element.
facet-option[]	[*string*]	Zero or more facet-options.
geo-option[]	[*string*]	Zero or more geo-options.
fragment-scope	*string*	Allowable values: • `properties` • `documents` (default) Set a local fragment scope for this constraint. The local `fragment-scope` overrides a global `fragment-scope`. For example, a `fragment-scope` of `properties` on a range constraint enables you to facet on a value stored in a property, even if you are searching over documents.
heatmap	heatmap-descriptor	Defines a heatmap to be used for faceting.

11.17 geo-path-constraint

```
{
  "path-index": path-expression,
  "facet-option": [ string ],
  "geo-option": [ string ],
  "fragment-scope": string,
  "heatmap": heatmap-descriptor
}
```

A geospatial constraint modelling a geospatial index with coordinates stored in a two elements. By default coordinates are stored as latitude,longitude points. To reverse the coordinate order, add the geo-option "long-lat-points" to the query options configuration. For details, see `cts:element-pair-geospatial-query`.

Key	Value	Description
path-index	path-expression	The XPath expression by which to constrain the search. The path expression is limited to the subset of XPath that can be used to define a path range index. For details, see Understanding Path Range Indexes in *Administrator's Guide*.
fragment-scope	*string*	Allowable values: • `properties` • `documents` (default) Set a local fragment scope for this constraint. The local `fragment-scope` overrides a global `fragment-scope`. For example, a `fragment-scope` of `properties` on a range constraint enables you to facet on a value stored in a property, even if you are searching over documents.
facet-option[]	[*string*]	Zero or more facet-options.
geo-option[]	[*string*]	Zero or more geo-options.
heatmap	heatmap-descriptor	Defines a heatmap to be used for faceting.

Example:

```
{ "options": {
    "constraint": [ {
        "name": "my-geo-path",
        "geo-path": {
          "path-index": {
            "text": "/ns1:a/ns2:b",
            "namespaces": {
              "ns1": "/my/ns-1",
              "ns2": "/my/ns-2"
            }
          }
        }
    } ]
} }
```

11.18 grammar

```
{
  "implict": string,
```

```
  "quotation": string,
  "starter": [ starter ],
  "joiner": [ joiner ]
}
```

Specifies a custom search grammar. For details, see Modifying and Extending the String Query Grammar in the *Search Developer's Guide*.

Key	Value	Description
implict	*string*	A `cts:query`, as a serialized string literal, to use by default to join two search terms together. By default, the Search API uses a `cts:and-query`, but you can change it to any `cts:query` with the implicit grammar option.
quotation	*string*	The string to use to indicate the start and end of a phrase.
starter	starter	Zero or more custom starters that denote special parsing for terms.
joiner	[joiner]	Zero or more custom joiners for joining two terms (cat AND dog) or for separating a constraint or operator from its value.

Example:

```
{
  "options": {
    "grammar": {
      "starter": [
        {
          "strength": 30,
          "apply": "grouping",
          "delimiter": ")",
          "label": "("
        }
      ],
      "joiner": [
        {
          "strength": 10,
          "apply": "infix",
          "element": "cts:or-query",
          "tokenize": "word",
          "label": "OR"
        }
      ],
      "quotation": "\"",
      "implicit": "<cts:and-query strength=\"20\"
xmlns=\"http:\/\/marklogic.com\/appservices\/search\"
```

```
xmlns:cts=\"http:\/\/marklogic.com\/cts\"\/>"
    }
  }
}
```

11.19 heatmap-descriptor

```
{
  "n": double,
  "s": double,
  "e": double,
  "w": double,
  "latdivs": number,
  "londivs": number
}
```

Models a two-dimensional grid used to categorize data along two dimenstions. Use with geospatial indexes.

Key	Value	Description
n	*double*	Required. The north coordinate.
s	*double*	Required. The south coordinate.
e	*double*	Required. The east coordinate.
w	*double*	Required. The west coordinate.
latdivs	*unsigned integer*	Required. The number of latitude divisions.
londivs	*unsigned integer*	Required. The number of longitude divisions.

11.20 joiner

```
{
  "apply": string,
  "ns": string,
  "at": string,
  "label": string,
  "strength": integer,
  "element": string,
  "options": string,
  "tokenize": string,
  "compare": string,
  "consume": integer
}
```

A joiner is a component of a custom string query grammar option that defines a binary operator that "joins" two string query expressions. The default grammar includes joiners such as `AND`, `OR`, `LT`, and colon (`:`).

Use the apply-at-ns pattern described in Search Customization Via Options and Extensions in *Search Developer's Guide* to define the XQuery library function that implements the joiner. The function must produce a cts:query element of the type named by `element`.

Key	Value	Description
label	*string*	Required. The joiner operator token. For example, the default grammar includes a joiner with the label `AND`, which enables queries like "`cat AND dog`".
apply	*string*	The localname of a function to use to apply the joiner.
ns	*string*	The namespace in which the `apply` function is defined, if using a user-defined function.
at	*string*	The module from which the `apply` function is imported, if using a user-defined function.
strength	*integer*	Required. The order of precedence of this joiner over others. Higher strength tokens or groups are processed before lower ones.
element	*string*	A `cts:query` element name (for example, `cts:and-query`).
options	*string*	A space-separated list of options that are passed through to the underlying `cts:query` constructor
tokenize	*string*	Allowed values: "word" or "default".
compare	*string*	The comparison operator used in range queries. Allowed values: LT, LE, GT, GE, NE
consume	*integer*	The number of tokens to consume when parsing the right operand of the joiner. For example, in the expression "A NEAR B", one token is consumed, but in the expression "A NEAR/3 B" two tokens are consumed (3 and B).

The following example defines `OR` as a joiner that produces a `cts:or-query` with precedence 10:

```
{
  "options": {
    "grammar": {
      "joiner": [
        {
          "strength": 10,
          "apply": "infix",
          "element": "cts:or-query",
          "tokenize": "word",
          "label": "OR"
        }
      ]
    }
  }
}
```

11.21 operator

```
{
  "name": string,
  "state": [
    {
      "name": string,
      "additional-query": string,
      "annotation": [ string ],
      "debug": boolean,
      "forest": unsigned long,
      "page-length": unsigned long,
      "quality-weight": double,
      "search-option": [ string ],
      "searchable-expression": path-expression,
      "sort-order": [ sort-order ],
      "transform-results": transform-results
    }
  ]
}
```

A named wrapper for one or more state elements, each representing a unique run-time configuration option. For example, if an operator with the name "sort" is defined, query text [sort:foo] will select the state child with the name "foo" at query runtime, using the option specified on that state element. Options affecting query parsing (such as `constraint`, `grammar`, `term`, `empty`) may not be configured via operators.

Key	Value	Description
name	*string*	Required. The name of this operator. The name must be unique across all operators and constraints
state[]	*array*	One or more state definitions.
state.name	*string*	Required. The name of this state.
state.additional-query	*string*	A query to perform when the operator is encountered.
state.annotation	[*string*]	Your comments. Annotations have no effect.
state.debug	*boolean*	Whether or not to enable debug logging. Default: false.
state.forest	*unsigned long*	A forest id.
state.page-length	*unsigned long*	The page length into which results are chunked.
state.quality-weight	*double*	Specifies a a weighting factor to use in the query. The default value is 1.0.
state.search-option[]	[*string*]	Zero or search options.
state.searchable-expression	path-expression	A query to perform when the operator is encountered.
state.sort-order[]	[sort-order]	Sort order rules to apply when the operator is encountered in a query.
state.transform-results	transform-results	Transformations to apply to the results produced by this operator.

The following example defines 2 operators, named “sort” and “page-length”. The “sort” operator has 2 states, “down” and “up”. The “page-length” operator has 2 states, “short” and “long”. This allows you, for example, to include a search term of the form `sort:up` to request results in ascending order.

```
{
  "options": {
    "operator": [
      {
        "name": "sort",
        "state": [
```

```
            {
              "name": "down",
              "sort-order": [{
                  "direction": "descending",
                  "score": null
                }]
            },
            {
              "name": "up",
              "sort-order": [{
                  "direction": "ascending",
                  "score": null
                }]
            }]
        },
        {
          "name": "page-length",
          "state": [
            {
              "name": "short",
              "page-length": 10
            },
            {
              "name": "long",
              "page-length": 50
            }]
        }]
    }
}
```

11.22 path-expression

```
{
  "text": string,
  "namespaces": { prefix-list }
}
```

Key	Value	Description
text	*string*	Required. An XPath expression. If the expression uses namespace prefixes, the prefixes must be defined in the `namespaces` key-value pair. Path elements with no prefix are assumed to be in no namespace.
namespaces	*prefix-list*	Optional. A list of `prefix:uri` pairs, defining any namespace prefixes used in the `text` property of this expression. See the example below.

The following example demonstrates using the path-expression “a:foo/b:bar/qux” as the definition of a searchable-expression:

```
{
  "options": {
    "searchable-expression": {
      "text": "\/a:foo\/b:bar\/qux",
      "namespaces": {
        "a": "http:\/\/a",
        "b": "http:\/\/b"
      }
    }
  }
}
```

11.23 properties-constraint

```
{ "properties": null }
```

Constrains a search to matches found in document properties.

Example:

```
{
  "options": {
    "constraint": [
      { "name": "just-props", "properties": null" }
    ]
  }
}
```

11.24 range-constraint

```
{
  "type": string,
  "element": element-descriptor,
  "attribute": attribute-descriptor,
  "field": field-descriptor,
  "json-key": string,
  "path-index": path-expression,
  "fragment-scope": string,
  "bucket": [ bucket ],
  "computed-bucket": [ computed-bucket ],
  "facet-option" : [ string ]
  "collation" : string,
  "facet": boolean,
  "range-option": [ string ]
}
```

Specifies an element, element attribute, field, JSON key, or XPath expression on which to constrain by range. The constraint must include one of `element`, `field`, `json-key`, or `path-index`. If there is an `element`, an `attribute` may also be included.

There must be a range index of the specified type (and collation for string range indexes) defined for the specified element, attribute, field, or path.

For range constraints with either bucket or computed-bucket specifications, for maximum performance and sortability, the buckets should be in a continuous order; if the order is not continuous (either ascending or descending), then the buckets are returned in the order specified, regardless of any sorting facet-option in the specification.

Key	Value	Description
type	*string*	Required. The XQuery type of the values in the range. There database configuration must include a range index with this value type.
element	element-descriptor	Defines an XML element to constrain by or the element containing `attribute`. If there is an `element`, there can also be an `attribute`. There must not be a `field`, `json-key`, or `path-index` specification. The database configuration must include a corresponding element or element attribute range index.
attribute	attribute-descriptor	Defines an XML element attribute to constrain by. There must be an accompanying element property. There must not be a `field`, `json-key`, or `path-index` specification. The database configuration must include a corresponding attribute range index.
field	field-descriptor	Defines field to constrain by. There must be an accompanying element property. There must not be a `element, attribute`, `json-key`, or `path-index` specification. The database configuration must include a corresponding field range index.
json-key	*string*	Defines JSON key to constrain by. There must be an accompanying element property. There must not be an `element, attribute`, `field`, or `path-index` specification. JSON key constraints apply only to JSON documents loaded using the MarkLogic REST API or the MarkLogic Java API.

Key	Value	Description
path-index	path-expression	Defines an XPath expression to constrain by. There must be an accompanying element property. There must not be a `element, attribute, json-key`, or `field` specification. The database configuration must include a corresponding path range index. The path expression is limited to the subset of XPath that can be used to define a path range index. For details, see Understanding Path Range Indexes in *Administrator's Guide*.
fragment-scope	*string*	Allowable values: • `properties` • `documents` (default) Set a local fragment scope for this constraint. The local fragment-scope overrides a global fragment-scope. For example, a `fragment-scope` of `properties` on a range constraint enables you to facet on a value stored in a property, even if you are searching over documents.
bucket[]	[bucket]	Zero or more named ranges of static values.
computed-bucket[]	[computed-bucket]	Zero or more named ranges of dynamic values.
facet-option[]	[*string*]	Zero or more facet-options.
collation	*string*	A collation URI, if `type` is `xs:string`.
facet	*boolean*	Whether or not to include facets.
range-option	[*string*]	Zero or more range-option settings.

Example:

```
{
  "options": {
    "constraint": [
      {
        "name": "size",
        "range": {
          "type": "xs:unsignedInt",
          "bucket": [
            {
              "name": "small",
              "lt": "5",
              "label": "Under 5"
```

```
            },
            {
              "name": "medium",
              "ge": "5",
              "lt": "12",
              "label": "5 to 12"
            },
            {
              "name": "large",
              "ge": "12",
              "label": "12+"
            }
          ],
          "element": {
            "name": "shoe",
            "ns": "http:\/\/implementation-specific.com"
          },
          "attribute": {
            "name": "size",
            "ns": ""
          }
        }
      }
    ]
  }
}
```

11.25 result-decorator

```
{
  "apply": string,
  "ns": string,
  "at": string
}
```

Defines a custom search result decorator XQuery function that is used to decorate each search `result` with additional information. The XQuery library module containing the function must already be installed in the App Server modules database at the XPath given by `at`.

Key	Value	Description
apply	*string*	The localname of a custom result decorator function. You must also specify `ns` and `at`.
ns	*string*	The namespace of the function in apply. This must match the namespace declaration of the XQuery library module containing the `apply` function.
at	*string*	The path to the XQuery library module containing the `apply` function.

The following example specifies the function `decorator` in the XQuery library module located at `/ext/my.domain/decorator.xqy`. assuming the module namespace is `http://marklogic.com/example/my-lib`.

```
{
  "options": {
    "result-decorator": {
      "apply": "decorator",
      "ns": "http://marklogic.com/example/my-lib",
      "at": "/ext/my.domain/decorator.xqy"
    }
  }
}
```

11.26 sort-order

```
{
  "element": element-descriptor,
  "attribute": attribute-descriptor,
  "field": field-descriptor,
  "json-key": string,
  "score": null,
  "annotation": [ string ],
  "type": string,
  "collation": string,
  "direction": string
}
```

Defines the sort order of an element, element attribute, field, or JSON property. A `sort-order` object must have one of a single `element` child, a single `score` child, a single `field` child, or a single `json-key` child.

If there is an `element` child it can optionally have an `attribute` sibling (to specify an attribute of the preceding element). Both the `element` and `attribute` elements must have `ns` and `name` attributes to specify the namespace and local-name of the specified element and attribute.

You should not apply `sort-order` to a field that has more than one included element.

If there is a score child, it specifies to sort by the score of the search result.

Key	Value	Description
element	element-descriptor	Identifies the element to which this sort order applies, or the element containing the attribute to which this sort order applies. If specifying only an element, you must set `type` to match the range index type of this element. If an element property is present, you cannot also specify `field` or `json-key`.
attribute	attribute-descriptor	Identifies the element attribute to which this sort order applies. You must set `type` to match the range index type of this attribute. If an `attribute` property is present, you cannot also specify `field` or `json-key`, and you must specify `element`.
field	field-descriptor	Identifies the field to which this sort order applies. Cannot be used with `element`, `attribute` or `json-key`.
json-key	*string*	Identifies the JSON property to which this sort order applies. This applies only to matches in JSON documents loaded using the MarkLogic REST API. Cannot be used with `element`, `attribute` or `field`.
score	null	Sort by the score of the search result. For details, see Relevance Scores: Understanding and Customizing in the *Search Developer's Guide*.
annotation[]	[*string*]	Your comments. Annotations have no effect.
type	*string*	If you are sorting by an element or an attribute, you must specify a type property with a value corresponding to the range index type of that element or attribute (for example, `xs:string`, `xs:dateTime`, and so on).
collation	*string*	A collation URI. Optionally specify a collation when-sort-order.type is xs:string; otherwise, use the collation of the query.
direction	*string*	Specify the sorting direction. Default: `descending`. Allowed values: • ascending • descending

For example, the following specifies a primary sort order using the element value for `my-element` (which needs a string range index with the specified collation), and a secondary sort order of score ascending:

```
{
  "options": {
    "sort-order": [
      {
        "direction": "descending",
        "type": "xs:string",
        "collation": "http://marklogic.com/collation/",
        "element": {
          "name": "my-element",
          "ns": "my-namespace",
        },
        "annotation": [ "some user comment can go here" ] },
      {
        "direction": "ascending",
        "score": null
      }
    ]
  }
```

11.27 starter

```
{
  "label": string,
  "apply": string,
  "ns": string,
  "at": string,
  "strength": integer,
  "element": string,
  "options": string,
  "tokenize": string,
  "delimiter": string
}
```

A `starter` defines a unary prefix operator or a pair of term grouping symbols. A `grammar` query option can contain 0 or more starters, in an array. Identify the XQuery parsing function using "apply", "at", and "ns", as described in Search Customization Via Options and Extensions in *Search Developer's Guide*. Use "element" to identify the type of `cts:query` element produced by the parsing function.

When defining a unary starter operator, put the operator token or symbol in the "label" key-value pair. When defining grouping symbols, put the group start symbol or token in the "label" key-value pair and the group end token or symbol in the "delimiter" key-value pair.

Use the "strength" key-value pair to set the precedence of this operator relative to other operators in the grammar. Higher strength operators are evaluated before lower strength ones.

Key	Value	Description
label	*string*	Required. The joiner operator token. For example, the default grammar includes a joiner with the label “AND”, which enables queries like “cat AND dog”.
apply	*string*	The local-name of a function to use to apply the starter.
ns	*string*	The namespace in which the `apply` function is defined, if using a user-defined function.
at	*string*	The module from which the `apply` function is imported, if using a user-defined function.
strength	*integer*	Required. The order of precedence of this starter over others. Higher strength tokens or groups are processed before lower ones.
element	*string*	A `cts:query` element name (for example, `cts:and-query`).
options	*string*	A space-separated list of options that are passed through to the underlying `cts:query` constructor
tokenize	*string*	Allowed values: “word” or “default”.
delimiter	*string*	The string to use as a delimiter for the starter.

The following example from the default grammar defines 2 starters, a unary minus (-) operator and parentheses (()) for grouping:

```
{
  "options": {
    "grammar": {
      "starter": [
        {
          "strength": 30,
          "apply": "grouping",
          "delimiter": ")",
          "label": "("
        },
        {
          "strength": 40,
          "apply": "prefix",
          "element": "cts:not-query",
          "label": "-"
```

```
        }
      ]
    }
  }
}
```

11.28 suggestion-source

```
{
  "ref": string,
  "suggestion-option": [ suggestion-option ]
  "annotation": [ string ]
  "range": range-constraint,
  "word": word-constraint,
  "value": value-constraint,
  "collection": collection-constraint,
  "properties": properties-constraint,
  "word-lexicon": word-lexicon
}
```

Specifies a constraint source that overrides a named constraint when using `search:suggest`. Suggestions are often used for type-ahead suggestions in a search user interface. For details, see Choose Suggestions With the suggestion-source Option in the *Search Developer's Guide*.

Specifying a `suggestion-source` can be useful when you use a named constraint for searching and facets, but you want to use a different source for type-ahead suggestions without re-parsing your search terms.

Each suggestion source must have a `ref` property corresponding to a named constraint (one `suggestion-source` object per named constraint) defined elsewhere in the options.

Use `suggestion-option` to override default behaviors of the constraint, such as sort order and case sensitivity. Use the constraint and lexicon related components such as `range`, `word`, and `collection` to override the constraint or lexicon named by `ref` with a new source.

> **Note:** The use of `word-lexicon` (the database-wide word lexicon) is not recommended as best practice; collection and range lexicons will yield the best performance.

A constraint defined within a `suggestion-source` can include a `collation` value, which specifies the collation of the value lexicon used during query evaluation. If no collation is specified, then the query uses the default collation for the context in which the query is evaluated.

If the `suggestion-source` specifies only `ref`, then no suggestions are generated when that constraint is applied. For example, the following options specifies that no suggestions should be generated for terms that would otherwise draw from the "color" constraint:

```
"suggestion-source": {"ref": "color"}
```

You can define also define a suggestion source for unqualified terms; for details, see “default-suggestion-source” on page 324.

A `suggestion-source` can contain the following properties. The `collection`, `range`, `word`, and `word-lexicon` properties are mutually exclusive.

Key	Value	Description
`ref`	*string*	Required. The name of a constraint defined elsewhere in the query options.
`suggestion-option`	[suggestion-option]	Query options to use in generating suggestions from this source. Each option is a string. The supported options depend on the constraint type.
`annotation`	[*string*]	Your comments. Annotations have no effect.
`range`	range-constraint	Defines a constraint that overrides the one named by `ref` when generating search suggestions.
`word`	word-constraint	Defines a constraint that overrides the one named by `ref` when generating search suggestions.
`collection`	collection-constraint	Defines a constraint that overrides the one named by `ref` when generating search suggestions.
`word-lexicon`	word-lexicon	Defines a constraint that overrides the one named by `ref` when generating search suggestions.

For example, the following options define 3 constraints (“color”, “price”, and “pets”), along with a `suggestion-source` option applicable to each. The suggestion source for “color” specifies suggestions matches should be case-sensitive. The suggestion source for “price” specifies that no suggestions should be generated from “price”. The suggestion source for “pets” specifies that suggestions should be drawn from an alternative source, rather than the collection defined by the original constraint.

```
{ "options": {
    "constraint": [
      { "name": "color",
        "word": { "json-key": "color" }
      },
      { "name":  "price",
```

```
          "range": {
            "type": "xs:unsignedInt",
            "element": { "ns": "", "name": "price" }
          }
        },
        { "name":  "pets",
          "collection": { "prefix": "/my/collection" }
        }
      ],
      "suggestion-source": [
        { "ref": "color", "suggestion-option": ["case-sensitive"] },
        { "ref": "price" },
        { "ref": "pets", "collection": { "prefix": "/my/subset" } }
      ]
} }
```

11.29 term-definition

Specifies handling of empty searches and controls options for how individual terms (that is, terms not associated with a constraint) will be represented when parsing the search.

```
{
  "term-option": [ string ],
  "default": {
    "ref": string
    "word": word-constraint,
    "value": value-constraint,
    "range": range-constraint,
    "annotation": [ string ],
  },
  "empty": {
    "apply": string,
    "ns": string,
    "at": string
  },
  "weight": double,
  "fragment-scope": string
  "apply": string,
  "ns": string,
  "at": string
}
```

Key	Value	Description
term-option[]	[*string*]	Zero or more term-options.
default	*object*	Specify a default property which determines special handling to all terms, so you can have all terms apply a set of rules such as query weighting, specifying a element or attribute value or word query to be used as a default, or specifying a constraint to be applied by default.
default.ref	*string*	A reference to a constraint defined elsewhere in the query options.
default.range	range-constraint	A range constraint to apply to individual terms without other constraints.
default.value	value-constraint	A value constraint to apply to individual terms without other constraints.
default.word	word-constraint	A word constraint to apply to individual terms without other constraints.
default.annotation	[*string*]	Your comments. Annotations have no effect.
apply	*string*	The name of a custom function to control individual term handling. If you specify a custom function name, you must also specify `term.ns` and `term.at`.
ns	*string*	Only required when using `term.apply`. The namespace of the function library module containing your function.
at	*string*	Only required when using `term.apply`. The path to the module containing your function.
empty	*object*	Use this property to control how empty searches (that is, the empty string passed into `search:search`) are resolved. The value of the `apply` child property specifies the behavior for empty searches:

Key	Value	Description
empty.apply	*string*	The behavior for empty searches. Allowed values: • `all-results`: return everything in the database • `no-results`: return nothing (default) • your own function name To specify your own function, create a function that returns a `cts:query` and specify the local-name of your function, along with the namespace of the function library module, and the path to the module. When specifying your own function, you must also specify `empty.ns` and `empty.at`.
empty.ns	*string*	Only required when `empty.apply` is a custom function. The namespace of the function library module containing your function.
empty.at	*string*	Only required when `empty.apply` is a custom function. The path to the module containing your function.
fragment-scope	*string*	Allowable values: • `properties` • `documents` (default) Set a local fragment scope for evaluating terms. The local `fragment-scope` overrides a global `fragment-scope`. However, this local scope is ignored if this `term` definition includes a constraint definition, constraint reference, or a custom function (via `apply`).
weight	*double*	A weighting factor to apply to individual terms. Default: 1.0.

An example of using a custom function custom code:

```
{
  "options": {
    "term": {
      "apply": "my-term-query",
      "ns": "http:\/\/custom\/search",
      "at": "\/lib\/custom.xqy",
      "empty": {
        "apply": "all-results"
      }
    }
  }
}
```

An example parsing to a weighted element word query on element "foo:bar":

```
{
  "options": {
    "term": {
      "default": {
        "word": {
          "element": {
            "ns": "foo",
            "name": "bar"
          },
          "weight": 2
        }
      },
      "empty": {
        "apply": "all-results"
      }
    }
  }
}
```

11.30 transform-results

```
{
  "apply": string,
  "ns": string,
  "at": string,
  "per-match-tokens": unsigned integer,
  "max-matches": unsigned integer,
  "max-snippet-chars": unsigned integer,
  "preferred-elements": [ element-descriptor ],
  "user-defined-param": "param-value"
}
```

Key	Value	Description
apply	*string*	The local-name of a function to use to process a search result for the snippet output. Default: “snippet”. There are three other built-in snippetting functions whose behavior you can affect: "raw" (return the whole node), "empty-snippet" (return an empty snippet), and "metadata-snippet" (return the snippet from the specified element in the properties document).
ns	*string*	The namespace in which the `apply` function is defined, if using a user-defined function.
at	*string*	The module from which the `apply` function is imported, if using a user-defined function.
per-match-tokens	*unsigned integer*	Usable with the builtin snippeting functions.
max-matches	*unsigned integer*	Usable with the builtin snippeting functions.
max-snippet-chars	*unsigned integer*	Usable with the builtin snippeting functions.
preferred-elements	[element-descriptor]	Usable with the builtin snippeting functions.
user-defined-param	*value*	Pass parameters to your snippeting function by including *`param-name`*`:value pairs`. For example, to pass in a value of 1 for the “limit” parameter of your function: “limit”:1.

For example, to modify the default snippeting function, “snippet”:

```
{
  "options": {
    "transform-results": {
      "apply": "snippet",
      "per-match-tokens": "30",
      "max-matches": "4",
      "max-snippet-chars": "200",
      "preferred-elements": ""
    }
  }
}
```

To define a custom transformation using your own snippeting function, “my-snippet” and pass it two parameters:

```
{
  "options": {
    "transform-results": {
      "apply": "my-snippet",
      "ns": "my-namespace",
      "at": "/my-snippet.xqy",
      "my-param1": "42",
      "my-param2": "abc"
    }
  }
}
```

11.31 value-constraint

```
{
  "json-key": string,
  "element": element-descriptor,
  "attribute": attribute-descriptor,
  "field": field-descriptor,
  "weight": double,
  "fragment-scope": string,
  "term-option": string
}
```

Defines elements, element attributes, fields, and JSON keys on which to constrain values. You cannot create facets from a value constraint.

Key	Value	Description
json-key	*string*	A JSON key on which to constrain queries. Only applicable to JSON documents ingested using the MarkLogic REST API or the MarkLogic JSON API.
element	element-descriptor	An element on which to constrain queries, or the element containing `attribute` when constraining by element attribute.
attribute	attribute-descriptor	An element attribute on which to constrain queries. The constraint must also contain an `element` if there is an `attribute`.

Key	Value	Description
field	field-descriptor	A field on which to constrain queries.
weight	*double*	Higher weights move search results up in the relevance order. The default is 1.0. The weight should be less than or equal to 64 and greater than or equal to -16 (between -16 and 64); weights greater than 64 will have the same effect as a weight of 64. Weights less than the absolute value of 0.0625 (between -0.0625 and 0.0625) are rounded to 0, which means that they do not contribute to the score.
fragment-scope	*string*	Allowable values: • properties • documents (default) Set a local fragment scope for this constraint. The local `fragment-scope` overrides a global `fragment-scope`. For example, a `fragment-scope` of `properties` on a range constraint enables you to facet on a value stored in a property, even if you are searching over documents.
term-option	*string*	Zero or more term-options.

Example:

```
{
  "options": {
    "constraint": [
      {
        "name": "my-value",
        "value": {
          "element": {
            "ns": "my-namespace",
            "name": "my-localname"
          }
        }
      }
    ]
  }
}
```

11.32 values-or-tuples

```
{
  "name": string,
  "uri": null,
  "annotation": [ string ],
```

```
    "collection": collection-constraint,
    "range": range-constraint,
    "field": field-descriptor,
    "geo-elem": geo-elem-constraint,
    "geo-elem-pair": geo-elem-pair-constraint,
    "geo-attr-pair": geo-attr-pair-constraint,
    "aggregate": {
      "apply": string,
      "udf": string
    },
    "values-option": [ string ]
  }
```

Key	Value	Description
name	*string*	Required. The name that identifies this values or tuples definition.
annotation	[*string*]	Your comments. Annotations have no effect.
uri	null	Specifies the database URI lexicon.
collection	collection-constraint	A collection constraint with which to extract values from the database collection lexicon.
range	range-constraint	A range constraint with which to extract values from a range index or value lexicon.
field	field-descriptor	A field constraint with which to extract values from a field index or value lexicon.
geo-elem	geo-elem-constraint	A geospatial constraint with which to extract values from a geospatial index.
geo-elem-pair	geo-elem-pair-constraint	A geospatial constraint with which to extract values from a geospatial index.
geo-attr-pair	geo-attr-pair-constraint	A geospatial constraint with which to extract values from a geospatial index.

Key	Value	Description
aggregate	*object*	Defines a builtin or user-defined aggregate function to apply to lexicon values or value co-occurrences.
aggregate.apply	*string*	Required. The name of a builtin aggregate or aggregate UDF.
aggregate.udf	*string*	The path to the native plugin library containing the implementation of the user-defined extension aggregate. Omit when using a builtin aggregate.
values-option[]	[*string*]	Zero or more values-options.

Example:

```
{
  "options": {
    "values": [
      {
        "name": "speaker",
        "range": {
          "type": "xs:string",
          "element": {
            "ns": "",
            "name": "SPEAKER"
          }
        }
      }
    ]
  }
}
```

11.33 word-constraint

```
{
  "json-key": string,
  "element": element-descriptor,
  "attribute": attribute-descriptor,
  "field": field-descriptor,
  "weight": double,
  "fragment-scope": string,
  "term-option": string,
  "collation": string
}
```

Constrains by a word query of JSON key, element, element attribute, or field. You cannot facet on word constraints.

Key	Value	Description
json-key	*string*	Constrain by JSON key. Mutually exclusive of `element`, `attribute`, and `field`.
element	element-descriptor	Constrain by JSON key. Mutually exclusive of `json-key` and `field`.
attribute	attribute-descriptor	Constrain by JSON key. Mutually exclusive of `json-key` and `field`. An `element` is also required.
field	field-descriptor	Constrain by JSON key. Mutually exclusive of `element`, `attribute`, and `field`.
weight	*double*	Higher weights move search results up in the relevance order. The default is 1.0. The weight should be less than or equal to the 64 and greater than or equal to -16 (between -16 and 64); weights greater than 64 will have the same effect as a weight of 64. Weights less than the absolute value of 0.0625 (between -0.0625 and 0.0625) are rounded to 0, which means that they do not contribute to the score.
fragment-scope	*string*	Allowable values: • `properties` • `documents` (default) Set a local fragment scope for this constraint. The local `fragment-scope` overrides a global `fragment-scope`. For example, a `fragment-scope` of `properties` on a range constraint enables you to facet on a value stored in a property, even if you are searching over documents.
term-option	*string*	Zero or more term-options.
collation	*string*	A collation URI.

11.34 word-lexicon

```
{
  "fragment-scope": string,
  "collation": string
}
```

Key	Value	Description
fragment-scope	*string*	Allowable values: • `properties` • `documents` (default)
collation	*string*	A collation URI. If no collation is specified, then the query uses default collation for the context in which the query is evaluated.

11.35 facet-options

Facet options are specified as properties of range and other constraints that can be used as a facet (any constraints except word, value, element-query, or property).

Legal values for a `facet-option` property are generally any option that can be passed into the underlying query or lexicon API. The following list enumerates the `facet-option` values, but be aware that some options are only available with some range types. For more detail on these options, see the documentation for the underlying range or lexicon APIs

- ascending
- descending
- empties
- any
- document
- properties
- locks
- frequency-order
- item-order
- fragment-frequency
- item-frequency
- type=type
- timezone=TZ
- limit=N
- sample=N
- truncate=N

- skip=N
- score-logtfidf
- score-logtf
- score-simple
- score-random
- checked
- unchecked
- concurrent
- map

For example:

```
{
  "options": {
    "constraint": [
      {
        "name": "color",
        "range": {
          "type": "xs:string",
          "facet-option": [
            "item-frequency"
          ],
          "element": {
            "ns": "",
            "name": "bodycolor"
          }
        }
      }
    ]
  }
}
```

11.36 term-options

Specifies the options passed into the query (for example, `case-insensitive`). By default, the query uses the same default options as the underlying cts:query constructors, and the defaults change based on your index configuration.

You can use term options in several places, including as a property of a term-definition and within many constraints. Within constraints, term options typically occur as property the type of constraint (for example, inside a word-constraint). You can use the `/config/query` service to validate a set of options.

Legal term options values are:

- case-sensitive

- case-insensitive
- diacritic-sensitive
- diacritic-insensitive
- punctuation-sensitive
- punctuation-insensitive
- whitespace-sensitive
- whitespace-insensitive
- stemmed
- unstemmed
- wildcarded
- unwilcarded
- exact
- lang=*iso639code*

For example

```
{
  "options": {
    "term": {
      "apply": "term",
      "empty": {
        "apply": "all-results"
      },
      "term-option":[
        "diacritic-insensitive",
        "unwildcarded"
      ]
    }
  }
}
```

11.37 values-options

Options that can be used in defining values and tuples. For a description of these options, see `cts:values` and `cts:values-tuples` in the *XQuery and XSLT Reference Guide*. Some options can only be used when defining a values configurations; others can only be used when defining tuples.

- ascending
- descending
- any
- document

- properties
- locks
- frequency-order
- item-order
- fragment-frequency
- item-frequency
- timezone=*TZ*
- limit=*N*
- skip=*N*
- sample=*N*
- truncate=*N*
- score-logtfidf
- score-logtf
- score-simple
- score-random
- score-zero
- checked
- unchecked
- concurrent
- map (for values only)
- ordered (for tuples only)
- proximity=*N* (for tuples only)

11.38 geo-options

Options that can be used when defining geo constraints. The options usable in a given geo constraint depend on the type of constraint. See the constraint for limitations. For an explanation of the options, see `cts:geospatial-element-query` or the equivalent for your range constraint type.

- `coordinate-system=`*wgs84*
- `units=`*miles*
- `boundaries-included`
- `boundaries-excluded`
- `boundaries-latitude-excluded`

- `boundaries-longitude-excluded`
- `boundaries-south-excluded`
- `boundaries-north-excluded`
- `boundaries-east-excluded`
- `boundaries-west-excluded`
- `boundaries-circle-excluded`
- `type=`*point*
- `type=`*long-lat-point*
- `score-function=zero` (default)
- `score-function=reciprocal`
- `score-function=linear`
- `slope-factor=`*number*
- `cached`
- `uncached`
- `synonym`

11.39 range-option

Options used to influence range query scoring. The following options are available.

- `score-function=zero` (default)
- `score-function=reciprocal`
- `score-function=linear`
- `slope-factor=`*number*
- `max-occurs=`*number*
- `min-occurs=`*number*
- `cached`
- `uncached`
- `synonym`

The `slope-factor` must be an `xs:double`.

For details, see Including a Range or Geospatial Query in Scoring in *Search Developer's Guide* and `cts:element-range-query` (or equivalent functions for other constraint types).

11.40 suggestion-option

Options usable when defining a search suggestion source, such as the `default-suggestion-source` or `suggestion-source` options. For the meaning of these options, refer to the underlying lexicon functions, such as `cts:values-match` or `cts:word-match`. For more details, see "suggestion-option" on page 365.

- case-sensitive
- case-insensitive
- diacritic-sensitive
- diacritic-insensitive
- ascending
- descending
- frequency-order
- item-order
- fragment-frequency
- item-frequency
- type=*type*
- timezone=*TZ*
- sample=*N*
- truncate=*N*
- score-logtfidf
- score-logtf
- score-simple
- score-random
- checked
- unchecked

12.0 Appendix B: XML Query Options Reference

This chapter defines the XML structure of query options (`search:options`) used by the search related features of the MarkLogic REST API and the MarkLogic Java API.

Query options can also be expressed in JSON. For details, see "Appendix A: JSON Query Options Reference" on page 308.

For more information, see Search Customization Using Query Options in the *Search Developer's Guide*.

12.1 <options>

The `<options>` node is the outer wrapper for query options. An options node can contain the following elements:

- <additional-query>
- <concurrency-level>
- <constraint>
- <debug>
- <default-suggestion-source>
- <extract-metadata>
- <forest>
- <fragment-scope>
- <grammar>
- <operator>
- <page-length>
- <quality-weight>
- <result-decorator>
- <return-constraints>
- <return-facets>
- <return-metrics>
- <return-plan>
- <return-qtext>
- <return-query>
- <return-results>
- <return-similar>

- <search-option>
- <searchable-expression>
- <sort-order>
- <suggestion-source>
- <term>
- <transform-results>
- <tuples>
- <values>

12.2 <additional-query>

An additional serialized `cts:query` node, which is combined using a `cts:and-query` with the query resulting from the query text. The query results are constrained by the specified `additional-query`, but any terms matching the `addtional-query` are not highlighted in the snippet result output. For example, the following options node constrains the results to the directory named /my/directory/:

```
<options xmlns="http://marklogic.com/appservices/search">
  <additional-query>
    <cts:directory-query xmlns:cts="http://marklogic.com/cts">
      <cts:uri>/my/directory/</cts:uri>
    </cts:directory-query>
  </additional-query>
</options>
```

If you have multiple `additional-query` options, they are combined using a `cts:and-query`.

12.3 <concurrency-level>

The maximum number of threads used to resolve facets. The default is 8, which specifies that at most 8 threads will be used concurrently to resolve facets. For example, the following specifies a concurrency level of 16:

```
<options xmlns="http://marklogic.com/appservices/search">
  <concurrency-level>16</concurrency-level>
</options>
```

12.4 <constraint>

The outer wrapper element for a constraint definition. Constraints are used to define facets which can be returned as part of the search results. The default is no defined constraints.

Each constraint element must have an `@name` attribute (required), which is the unique name of this constraint. The name can then be used in the search grammar to specify the constraint. A constraint name must be unique within the containing query options and must not contain whitespace.

The constraint element can have zero or more of the following elements:

- <value>
- <word>
- <collection>
- <range>
- <container>
- <element-query>
- <properties>
- <geo-*>
- <custom>
- <term-option>
- <facet-option>
- <range-option>
- <geo-option>

12.4.1 <value>

Specifies element or attribute values on which to constrain. You cannot create facets from a value constraint. For example:

```
<constraint name="my-value">
    <value>
      <element ns="my-namespace" name="my-localname"/>
    </value>
</constraint>
<constraint name="my-attribute-value">
    <value>
      <attribute ns="" name="my-attribute"/>
      <element ns="my-namespace" name="my-localname"/>
    </value>
</constraint>
<constraint name="fieldvalue">
    <value>
      <field name="fieldvalue"/>
      <weight>2.0</weight>
    </value>
</constraint>
```

12.4.2 <word>

Specifies the element, attribute, or field on which to constrain by word. You cannot create facets from a word constraint. For example:

```
<constraint name="name">
    <word>
      <element ns="http://widgets-r-us.com" name="name"/>
      <weight>2.0</weight>
    </word>
</constraint>
<constraint name="description">
    <word>
      <field name="my-field"/>
    </word>
</constraint>
```

12.4.3 <collection>

Specifies the collection on which to constrain. For example:

```
<constraint name="my-collection">
    <collection prefix="http://server.com/my-collection/"/>
</constraint>
```

12.4.4 <range>

Specifies the element or attribute on which to constrain by range. There must be a range index of the specified type (and collation for string range indexes) defined for the specified element or attribute.

Each range element with a type attribute child, an optional collation attribute child (for string range indexes), an element child, an optional attribute child (for attribute ranges), and may have one or more computed-bucket children and/or one or more facet-option children (to pass options to the underlying lexicon apis). If it is a field range constraint, then the range element has a field child element with a name attribute. If it is a path-index range constraint, then the range element has a path-index child element with any namespaces defined in xmlns attributes. For example:

```
<options xmlns="http://marklogic.com/appservices/search">
 <constraint name="made">
    <range type="xs:dateTime"><!-- requires a dateTime range index -->
    <element ns="http://example.com" name="manufactured"/>
    <attribute ns="" name="date"/>
    <computed-bucket name="today" ge="P0D" lt="P1D"
            anchor="now">Today</computed-bucket>
    <computed-bucket name="30-days" ge="-P30D" lt="P1D"
            anchor="now">Last 30 days</computed-bucket>
    <computed-bucket name="60-days" ge="-P60D" lt="P1D"
            anchor="now">Last 60 Days</computed-bucket>
    <computed-bucket name="year" ge="-P1Y" lt="P1D"
            anchor="now">Last Year</computed-bucket>
  </range>
```

```
</constraint>
<constraint name="color">
  <range type="xs:string">
    <element ns="" name="bodycolor"/>
  </range>
</constraint>
<constraint name="name">
  <range type="xs:string" collation="http://marklogic.com/collation/">
    <field name="my-field-name"/>
  </range>
</constraint>
<constraint name="color-facet">
   <range type="xs:string" facet="true">
     <element ns="" name="bodycolor"/>
     <!-- the facet-option values are passed directly to the
          underlying lexicon calls -->
     <facet-option>frequency-order</facet-option>
     <facet-option>descending</facet-option>
   </range>
</constraint>
<constraint name="fieldrange">
   <range type="xs:string"
           collation="http://marklogic.com/collation/">
      <field name="fieldrange"/>
   </range>
</constraint>
<constraint name="fieldrangebucket">
  <range type="xs:dateTime">
    <facet-option>ascending</facet-option>
    <field name="fieldrangebucket"/>
    <computed-bucket name="older" lt="-P365D" anchor="now">
      Older than 1 years
    </computed-bucket>
    <computed-bucket name="year" ge="-P365D" lt="-P30D" anchor="now">
      1 month to 1 year ago
    </computed-bucket>
    <computed-bucket name="month" ge="-P30D" lt="-P7D" anchor="now">
      7 to 30 days ago
    </computed-bucket>
    <computed-bucket name="week" ge="-P7D" lt="-P1D" anchor="now">
      1 to 7 days ago
    </computed-bucket>
    <computed-bucket name="today" ge="-P1D" lt="P0D" anchor="now">
      Today
    </computed-bucket>
    <computed-bucket name="future" ge="P0D" anchor="now">
      Future
    </computed-bucket>
   </range>
</constraint>
<constraint name="year">
  <range type="xs:gYear" facet="true">
    <path-index xmlns:my="http://example.com">
      /publication/my:meta/my:year
```

```
      </path-index>
      <!--uses same options structure, options as element,
          element-attribute or field based range constraints -->
    </range>
  </constraint>
</options>
```

For range constraints with either bucket or computed-bucket specifications, for maximum performance and sortability, the buckets should be in a continuous order; if the order is not continuous (either ascending or descending), then the buckets are returned in the order specified, regardless of any sorting facet-option in the specification.

12.4.5 <container>

Specifies a constraint that restricts a search to a specified XML element or JSON key container. You cannot create facets from a container constraint.

To specify an element container, include an `<element/>` child. For example:

```
<options xmlns="http://marklogic.com/appservices/search">
  <constraint name="sample-container-constraint">
    <container>
      <element name="title" ns="http://my/namespace" />
    </container>
  </constraint>
</options>
```

To specify a JSON key container, include a `<json-key>` child. For example:

```
<options xmlns="http://marklogic.com/appservices/search">
  <constraint name="sample-container-constraint">
    <container>
      <json-key>key-name</json-key>
    </container>
  </constraint>
</options>
```

You can include a `<fragment-scope/>` child to limit the scope of the constraint to documents or properties. For details, see “<fragment-scope>” on page 380.

12.4.6 <element-query>

Note: This option is deprecated. Use a <container> constraint instead.

Specifies a constraint that restricts the search to the specified element. You cannot create facets from an element-query constraint.

```
<options xmlns="http://marklogic.com/appservices/search">
  <constraint name="sample-element-constraint">
    <element-query name="title" ns="http://my/namespace" />
```

```
  </constraint>
</options>
```

12.4.7 <properties>

Specifies a constraint that finds matches on the corresponding properties documents. You cannot create facets from a properties constraint.

```
<options xmlns="http://marklogic.com/appservices/search">
  <constraint name="sample-property-constraint">
    <properties />
  </constraint>
</options>
```

For an additional example, see the second example below.

12.4.8 <geo-*>

Specifies a geospatial constraint. Must be one of the following elements: `<geo-attr-pair>`, `<geo-elem-pair>`, `<geo-elem>`, or `<geo-path>`. You can use <geo-option> elements to further customize the constraint. You can use <fragment-scope> to control the scope of the search.

To use a geospatial contstraint as a facet, include a `<heatmap/>` child element. If a `<heatmap/>` is present, the constraint can also contain <facet-option> elements. For example:

```
<constraint name="my-geo-attr-pair">
    <!-- Uses cts:element-attribute-pair-geospatial-query, and
    cts:element-attribute-pair-geospatial-boxes for the
    heatmap facet.  -->
  <geo-attr-pair>
    <heatmap s="23.2" w="-118.3" n="23.3" e="-118.2"
             latdivs="4" londivs="4"/>
    <facet-option>empties</facet-option>
    <parent ns="ns1" name="elem1"/>
    <lat ns="ns2" name="attr2"/>
    <lon ns="ns3" name="attr3"/>
  </geo-attr-pair>
</constraint>
<constraint name="my-geo-elem-pair">
    <!-- Uses cts:element-pair-geospatial-query, and
    cts:element-pair-geospatial-boxes for the
    heatmap facet (not shown).  -->
  <geo-elem-pair>
    <parent ns="ns1" name="elem2"/>
    <lat ns="ns2" name="attr2"/>
    <lon ns="ns3" name="attr3"/>
  </geo-elem-pair>
</constraint>
<constraint name="my-geo-elem">
    <!-- Uses cts:element-geospatial-query, and
    cts:element-geospatial-boxes for the
    heatmap facet (not shown).  -->
```

```
  <geo-elem>
    <element ns="ns1" name="elem3"/>
    <geo-option>type=long-lat-points</geo-option>
  </geo-elem>
</constraint>
<constraint name="my-geo-elem-child">
    <!-- Uses cts:element-child-geospatial-query, and
    cts:element-geospatial-boxes for the
    heatmap facet (not shown).  -->
  <geo-elem>
    <parent ns="ns1" name="elem4"/>
    <element ns="ns1" name="elem5"/>
  </geo-elem>
</constraint>
<constraint name="my-geo-path">
    <!-- Uses cts:element-child-geospatial-query, and
    cts:element-geospatial-boxes for the
    heatmap facet (not shown).  -->
  <geo-path>
    <path-index>/a/b</path-index>
  </geo-elem>
</constraint>
```

Note that when specifying coordinates using a geospatial constraint, you must put any coordinate values that contain the negative sign (-) in double-quotes, otherwise the search API will interpret the negative sign as the negation operator (when using the default grammar). For example, to specify a point using a geospatial constraint, specify qtext of the form my-geo-elem-child:"-12, 5" in your search string.

The `path-index` XPath expression in a `geo-path` constraint is limited to the subset of XPath that can be used to define a path range index. For details, see Understanding Path Range Indexes in *Administrator's Guide*.

12.4.9 <custom>

Specifies a custom constraint along with the name of the function implementations used to evaluate the custom constraint. For example:

```
<constraint name="my-custom">
  <custom facet="true">
    <parse apply="my-parse-function"
        ns="my-function-namespace" at="path-to-module.xqy"/>
    <start-facet apply="my-start-function"
        ns="my-function-namespace" at="path-to-module.xqy"/>
    <finish-facet apply="my-finish-function"
        ns="my-function-namespace" at="path-to-module.xqy"/>
  </custom>
</constraint>
<!--  The start-facet and finish-facet elements can be omitted if
      facet="false". When facet="true", the start-facet can be ommitted
      if you do not run in concurrent mode ("concurrent" option on the
```

```
      lexicon functions).
-->
```

12.4.10 <term-option>

Specifies the options passed into the search (for example, case-insensitive). There can be zero or more `term-option` elements. By default, the search uses the same default options as the underlying `cts:query` constructors, and the defaults change based on your index configuration. You can use `term-option` elements in several places, including as a child of the term element and within many constraint elements (except for `element-query` constraints). Within constraints, `term-option` specifications typically occur as a child of the type of constraint (for example, in a word constraint, as a child of word. Use `search:check-options` to make sure the usage is legal.

Legal term option values are:

- case-sensitive
- case-insensitive
- diacritic-sensitive
- diacritic-insensitive
- punctuation-sensitive
- punctuation-insensitive
- whitespace-sensitive
- whitespace-insensitive
- stemmed
- unstemmed
- wildcarded
- unwilcarded
- exact
- lang=*iso639code*

For example:

```
<term-option>diacritic-insensitive</term-option>
```

Or, for example, within in a word constraint:

```
<constraint name="name"
  xmlns="http://marklogic.com/appservices/search">
  <word>
    <element ns="http://widgets-r-us.com" name="name"/>
    <term-option>case-sensitive</term-option>
```

```
    </word>
  </constraint>
```

12.4.11 <facet-option>

Specifies options for facets, and is specified as a child of any range constraint or other constraint that can be used as a facet (any constraints except word, value, element-query, or property).

Legal values for a `facet-option` element are generally any option that can be passed into the underlying `cts:range-query` or lexicon API. The following list enumerates the facet-option values, but be aware that some options are only available with some range types. For more detail on these options, see the documentation for the underlying range or lexicon APIs.

- ascending
- descending
- empties
- any
- document
- properties
- locks
- frequency-order
- item-order
- fragment-frequency
- item-frequency
- type=type
- timezone=TZ
- limit=N
- sample=N
- truncate=N
- skip=N
- score-logtfidf
- score-logtf
- score-simple
- score-random
- checked
- unchecked

- concurrent
- map

For example:

```
<constraint name="color"
xmlns="http://marklogic.com/appservices/search">
   <range type="xs:string">
     <element ns="" name="bodycolor"/>
     <facet-option>item-frequency</facet-option>
   </range>
</constraint>
```

12.4.12 <range-option>

Options used to influence range query scoring. The following options are available.

- `score-function=zero` (default)
- `score-function=reciprocal`
- `score-function=linear`
- `slope-factor=`*number*
- `max-occurs=`*number*
- `min-occurs=`*number*
- `cached`
- `uncached`
- `synonym`

The `slope-factor` must be an `xs:double`.

For details, see Including a Range or Geospatial Query in Scoring in the *Search Developer's Guide*.

12.4.13 <geo-option>

Options that can be used when defining geospatial constraints. The options usable in a given constraint depend on the type of constraint. See the constraint for limitations. For an explanation of the options, see `cts:geospatial-element-query` or the equivalent for your range constraint type.

- coordinate-system=*wgs84*
- units=*miles*
- boundaries-included
- boundaries-excluded

- boundaries-latitude-excluded
- boundaries-longitude-excluded
- boundaries-south-excluded
- boundaries-north-excluded
- boundaries-east-excluded
- boundaries-west-excluded
- boundaries-circle-excluded
- cached
- uncached
- type=*point*
- type=*long-lat-point*
- any <range-option>
- score-function=zero (default)
- score-function=reciprocal
- score-function=linear
- slope-factor=*number*
- synonym

12.5 <debug>

Activates debugging mode. Additional report elements will be present in the output. Set to true to activate. The default is false.

12.6 <default-suggestion-source>

Defines the content to be used as the default source of suggestions (see `search:suggest`). The source may be expressed as a reference to a named constraint (`@ref`) defined elsewhere in the options, or as a `collection`, `value`, `word` or `word-lexicon` element. For details, see default-suggestion-source Option in the *Search Developer's Guide*.

> **Note:** The use of `word-lexicon` (the database-wide word lexicon) is not recommended as best practice; collection and range lexicons will yield the best performance.

Each `default-suggestion-source` constraint element can optionally have an `@collation` attribute, which specifies the collation of the value lexicon used during query evaluation. If no collation is specified, then the query uses default collation for the context in which the query is evaluated.

Include one or more `suggestion-option` elements to specify behaviors like sort order when defining a constraint within the `default-suggestion-source`. For a list of options, see "suggestion-option" on page 365.

The following example draws suggestions for unconstrained terms from the range constraint named "animal".

```
<search:options xmlns="http://marklogic.com/appservices/search">
 <constraint name="animal">
   <range type="xs:string">
      <element name="beast"/>
   </range>
 </constraint>
 <default-suggestion-source ref="animal" />
</search:options>
```

The following example draws suggestions for unconstrained terms from the range constraint defined within the `default-suggestion-source`.

```
<search:options xmlns="http://marklogic.com/appservices/search">
 <default-suggestion-source ref="animal">
   <range type="xs:string">
     <element name="beast"/>
   </range>
   <suggestion-option>descending</suggestion-option>
 </default-suggestion-source>
</search:options>
```

The `default-suggestion-source` element can have zero or more of the following child elements:

- <collection>
- <range>
- <word>
- <word-lexicon>

12.6.1 <collection>

Specifies using the collection lexicon for suggestions. For example:

```
<default-suggestion-source>
        <collection/>
</default-suggestion-source>
```

12.6.2 <range>

Specifies the element or attribute lexicon to use for suggestions. For example:

```
<default-suggestion-source>
  <range type="xs:string">
```

```
    <element ns="my-namespace" name="my-localname"/>
    <attribute ns="" name="my-attribute"/>
  </range>
</default-suggestion-source>
```

12.6.3 <word>

Specifies using a constrained word lexicon for suggestions. This option might not scale well for a large database. For example:

```
<default-suggestion-source>
  <word>
    <field name="my-field"/>
  </word>
</default-suggestion-source>
```

12.6.4 <word-lexicon>

Specifies using the database-wide word lexicon for suggestions. This option might not scale well for a large database. For example:

```
<default-suggestion-source>
  <word-lexicon/>
</default-suggestion-source>
```

12.7 <extract-metadata>

Specifies data to return in the search:metadata section of the search:response output. You can specify element or attribute content from the search hits to be returned or values from any constraints in your search.

```
<search:extract-metadata>
           <!-- qnames specify elements or attributes in document
                content; multiples will be returned if they exist -->
   <search:qname elem-ns="" elem-name="bar"/>
   <search:qname elem-ns="http://example.com" elem-name="baz"
                 attr-ns="http://ick" attr-name="id"/>
   <search:qname elem-ns="http://example.com"
                 elem-name="blech" attr-ns="" attr-name="size"/>
           <!-- constraint-values extract values from the index
                associated with the constraint -->
   <search:constraint-value ref="idx"/>
   <search:constraint-value ref="id"/>
   <search:qname elem-ns="http://example.com" elem-name="foo"/>
</search:extract-metadata>
```

12.8 <forest>

A single forest ID to pass into cts:search. To specify multiple forests, use multiple forest elements in the options node.

The value must be an `xs:unsignedLong` type.

12.9 <fragment-scope>

Controls the scope over which to search. Valid values are documents and properties, and the default value is documents.

You can specify fragment-scope as a child of `<options>`, and that sets a global fragment scope which applies both to what the search returns (that is, if it returns results from document fragments or from property fragments) and the global scope is inherited by any constraints that do not explicitly override the fragment-scope.

You can also specify fragment-scope as a child of `<range>`, `<value>`, or `<word>` constraints. Specifying as a child of one of these constraints sets a local fragment scope for that constraint. The local fragment-scope overrides a global fragment-scope. For example, a fragment-scope of properties on a range constraint enables you to facet on a value stored in a property, even if you are searching over documents. For example:

```
<options xmlns="http://marklogic.com/appservices/search">
<fragment-scope>documents</fragment-scope>
  <constraint name="last-modified">
    <range type="xs:dateTime">
      <element ns="http://marklogic.com/xdmp/property"
               name="last-modified"/>
      <fragment-scope>properties</fragment-scope>
    </range>
  </constraint>
  <debug>true</debug>
</options>
```

12.10 <grammar>

Wrapper element for grammar definition. The default grammar defines "Google-style" parsing.

The grammar element has a `quotation` element that specifies the quotation character with which to surround phrases. The text between the quotation characters is treated as a phrase. You cannot specify a search that includes the quotation character; for example, to specify a search that includes the double quotation character (the default quotation character), modify your grammar to use a different quotation character.

The grammar element can have 0 or more `joiner` elements and 0 or more `starter` elements.

The grammar element should contain at least one of the following elements. If the grammar element is present but empty, then the grammar does nothing, and the search is parsed according to the term option.

- <implicit>
- <joiner>

- <starter>

12.10.1 <implicit>

Specifies the `cts:query` to use to implicitly combine terms when no joiner is specified in the query text. For example, how to combine term in a query like “cat dog”. Use this to modify the implicit search grammar (for example, if you want the default search to be a cts:or-query instead of a cts:and-query).

The following is an example of a options element with an implicit specification for a cts:or-query:

```
<options xmlns="http://marklogic.com/appservices/search">
  <grammar>
    <implicit>
      <cts:or-query strength="10"
         xmlns:cts="http://marklogic.com/cts"/>
    </implicit>
  </grammar>
</options>
```

12.10.2 <joiner>

```
<joiner xmlns="http://marklogic.com/appservices/search"
         strength="number" element="query_type"
         apply="func_name" at="lib_module" ns="namespace"
         options="space_separated_options" >
  starter_symbol_or_token
</starter>
```

A joiner defines a binary operator that “joins” two query expressions. Examples of joiners in the default grammar include `AND`, `OR`, `LT`, and colon (:). A `grammar` element can contain 0 or more `joiner` elements.

A joiner can contain the following attributes and elements:

Attribute or Element	Description
`@strength`	Required. The precedence of this operator, expressed as an integer. Higher strength operators are evaluated before lower valued ones.
`@apply`	Required. The localname of the function to use for processing the joiner operation.
`@at`	The XQuery library module containing the `apply` function implementation, if using a user-defined function.
`@ns`	The namespace in which the `apply` function is defined, if using a user-defined function.
`@element`	The `cts:query` element type produced by parsing this joiner. For example, the default grammar defines an `AND` joiner which produces a `cts:and-query`.
`text()`	Required. The token or symbol that identifies this starter. When defining grouping operators, the token or symbol that marks the beginning of a group.
`@options`	A space separated list of query options to pass into the `@apply` function.
`@tokenize`	Define show to tokenize the expression. Allowed values: `word` or `default`.
`@compare`	The comparison operation to perform when defining a relational operator. Allowed values: `LT`, `LE`, `GT`, `GE`, `NE`, `EQ`.
`@consume`	The number of additional terms to consume when parsing this operator. For example, the `near/` operator defined by the default grammar consumes 2 terms when parsing a query like `foo NEAR/8 bar`: The count (8) and "bar".

The following examples from the default grammar define the `AND` and `LT` operators, respectively.

```
<joiner strength="20" apply="infix" element="cts:and-query"
        tokenize="word">AND</joiner>
<joiner strength="50" apply="constraint" compare="LT"
        tokenize="word">LT</joiner>
```

12.10.3 <starter>

```
<starter xmlns="http://marklogic.com/appservices/search"
         strength="number" element="query_type"
         apply="func_name" at="lib_module" ns="namespace"
         delimiter="token" options="space_separated_options" >
  starter_symbol_or_token
</starter>
```

A `<starter/>` defines a unary prefix operator or a pair of term grouping symbols. A `<grammar/>` element can contain 0 or more `<starter/>` elements. Identify the XQuery parsing function using the `apply`, `at`, and `ns` attributes, as described in Search Customization Via Options and Extensions in *Search Developer's Guide*. Use the `element` attribute to identify the type of `cts:query` element produced by the parsing function.

When defining a unary starter operator, put the operator token or symbol in the text node. When defining grouping symbols, put the group start symbol or token in the text node and the group end token or symbol in the `delimiter` attribute.

Use the `strength` attribute to set the precedence of this operator relative to other operators in the `<grammar/>`. Higher strength operators are evaluated before lower strength ones.

Attribute or Element	Description
`@strength`	Required. The precedence of this operator, expressed as an integer. Higher strength operators are evaluated before lower valued ones.
`@apply`	Required. The localname of the function to use for parsing the starter sub-expression.
`@at`	The module containing the `apply` function implementation, if using a user-defined function.
`@ns`	The namespace in which the `apply` function is defined, if using a user-defined function.
`@element`	The `cts:query` element type produced by parsing this starter. For example, the default grammar defines a negation starter using the minus sign, which produces a `cts:not-query`.
`@delimiter`	When defining a group operator, the token or symbol that marks the end of a group.
`text()`	Required. The token or symbol that identifies this starter. When defining grouping operators, the token or symbol that marks the beginning of a group.
`@options`	A space separated list of query options to pass into the `@apply` function.

The following example from the default grammar defines 2 starters, a unary minus (-) operator and parentheses (()) for grouping:

```
<grammar xmlns="http://marklogic.com/appservices/search">
  <starter streznght="30" apply="grouping" delimiter=")">(</starter>
  <starter strength="40" apply="prefix"
```

```
      element="cts:not-query">-</starter>
    ...
</grammar>
```

12.11 <operator>

A named wrapper for one or more state elements, each representing a unique run-time configuration option. For example, if an operator with the name "sort" is defined, query text [sort:foo] will select the state child with the name "foo" at query runtime, using the option specified on that state element. Options affecting query parsing (such as constraint, grammar, term, empty) may not be configured via operators.

An operator element can have one or more state elements. Each state element can have one of the following elements:

- additional-query
- debug
- forest
- page-length
- quality-weight
- searchable-expression
- sort-order
- transform-results

In the following example, a search for special:hello constrains the search by the "hello world" query, and a search for special:forest constrains the search to the forest names "my-forest".

```
<operator name="special">
  <state name="hello">
    <additional-query>{cts:word-query("hello world")}
    </additional-query>
  </state>
  <state name="forest">
    <forest>{xdmp:forest("my-forest")}</forest>
  </state>
</operator>
```

12.12 <result-decorator>

Specifies a custom search result decorator XQuery function that is used to decorate each search `result` with additional information.

The following example specifies the function `decorator` in the XQuery library module located at `/ext/my.domain/decorator.xqy`. assuming the module namespace is `http://marklogic.com/example/my-lib`.

```
<options xmlns="http://marklogic.com/appservices/search">
  <result-decorator apply="decorator"
      ns="http://marklogic.com/example/my-lib"
      at="/ext/my.domain/decorator.xqy"/>
</options>
```

The XQuery library module containing the function must be installed under the App Server root or in the modules database at the XPath given by `@at`.

12.13 <page-length>

Specifies the number of results per page. The default value is 10.

The value must be an `xs:unsignedInt type`.

12.14 <quality-weight>

Specifies a a weighting factor to use in the query. The default value is 1.0.

The value must be an `xs:double type`.

12.15 <return-constraints>

Include original constraint definitions in the results. The default is false.

The value must be an `xs:boolean` type.

12.16 <return-facets>

Include resolved facets in the results. The default is true.

The value must be an `xs:boolean` type.

12.17 <return-metrics>

Include statistics in the results. The default is true.

The value must be an `xs:boolean` type.

12.18 <return-plan>

Include xdmp:plan output in the results. The default is false. The output is useful in examining the evaluation plan of the search.

The value must be an `xs:boolean` type.

12.19 <return-qtext>

Include the original query text in the results. The default is true.

The value must be an `xs:boolean` type.

12.20 <return-query>

Include the XML query representation in the results. The default is false.

The value must be an `xs:boolean` type.

12.21 <return-results>

Include search results in the output. (Use transform-results to specify how each result should be formatted.) The default is true.

The value must be an `xs:boolean` type.

12.22 <return-similar>

Include with each search result a list of URLs of similar documents in the database. The default is false.

The value must be an `xs:boolean` type.

12.23 <search-option>

For advanced users, a single option to be passed in with `cts:search` calls (for example, filtered, unfiltered, score-logtfidf, and so on). To pass in multiple options, specify multiple search-options elements in the options node. The default is no additional options. For example:

```
<search-option>unfiltered</search-option>
<search-option>score-logtf</search-option>
```

12.24 <searchable-expression>

An expression to be searched. Whatever expression is specified is returned from the search. For example, if you specify `//p`, then `p` elements that match the search criteris are returned. The expression must be an inline fully searchable XPath expression, and all necessary namespaces must be declared using xmlns attributes. For example:

```
<searchable-expression xmlns:ex="http:example.com"
    xmlns:com="http://company.com">/ex:orders/com:company
</searchable-expression>
```

The default value is `fn:collection()`, which searches all documents in the database.

Note: This option does not affect facet results. To constrain facet results, use <additional-query>.

12.25 <sort-order>

Set the default sort order. The first such element is the primary sort order, the second secondary sort order, and so on. The default is to sort by score, descending. Note that the default scoring algorithm can be set just like any other option with the option named `search-option`. If you are sorting by an element or an attribute, you must specify a type attribute with a value corresponding to the range index type of that element or attribute (for example, `xs:string`, `xs:dateTime`, and so on). If the corresponding range index is of type `xs:string`, then you can optionally specify a `collation` attribute (otherwise the collation of the query is used). To change the sorting direction, specify an optional `direction` attribute with a value of `descending` (the default) or `ascending`.

You should not apply `sort-order` to a field that has more than one included element.

The `sort-order` element must have one of a single `element` child, a single `score` child, a single `field` child, or a single `json-key` child. If there is a `score` child, it specifies to sort by the score of the search result. If there is an `element` child it can optionally have an `attribute` sibling (to specify an attribute of the preceding element). Both the `element` and `attribute` elements must have `ns` and `name` attributes to specify the namespace and local-name of the specified element and attribute. Additionally, the `sort-order` element can have 0 or more `annotation` elements (to add comments, for example).

For example, the following specifies a primary sort order using the element value for `my-element` (which needs a string range index with the specified collation), and a secondary sort order of score ascending:

```
<sort-order type="xs:string"
collation="http://marklogic.com/collation/"
            direction="ascending">
  <element ns="my-namespace" name="my-element"/>
  <annotation>some user comment can go here</annotation>
</sort-order>
<sort-order direction="ascending">
  <score/>
</sort-order>
```

12.26 <suggestion-source>

Specifies a constraint source that overrides a named constraint when using `search:suggest`. Suggestions are often used for type-ahead suggestions in a search user interface. For details, see Choose Suggestions With the suggestion-source Option in the *Search Developer's Guide*.

Specifying a `suggestion-source` can be useful when you use a named constraint for searching and facets, but you want to use a different source for type-ahead suggestions without re-parsing your search terms.

Each suggestion source must have a `ref` attribute corresponding to a named constraint (one `suggestion-source` object per named constraint) defined elsewhere in the options.

Use `suggestion-option` to override default behaviors of the constraint, such as sort order and case sensitivity. Use the constraint and lexicon related components such as `range`, `word`, and `collection` to override the constraint or lexicon named by `ref` with a new source.

> **Note:** The use of `word-lexicon` (the database-wide word lexicon) is not recommended as best practice; collection and range lexicons will yield the best performance.

A constraint defined within a `suggestion-source` can include a `collation` value, which specifies the collation of the value lexicon used during query evaluation. If no collation is specified, then the query uses the default collation for the context in which the query is evaluated.

If the `suggestion-source` specifies only `ref`, then no suggestions are generated when that constraint is applied. For example, the following options specifies that no suggestions should be generated for terms that would otherwise draw from the "color" constraint:

```
<suggestion-source ref="color" />
```

You can define also define a suggestion source for unqualified terms; for details, see "<default-suggestion-source>" on page 377.

A `suggestion-source` can contain the following elements and attributes. The `collection`, `range`, `word`, and `word-lexicon` elements are mutually exclusive.

Key	Description
`@ref`	Required. The name of a constraint defined elsewhere in the query options.
`suggestion-option`	A query option to use in generating suggestions. This element can appear multiple times. For a list of supported options, see "suggestion-option" on page 365.
`annotation`	Your comments. Annotations have no effect.

Key	Description
range	Defines a constraint that overrides the one named by ref when generating search suggestions. For details, see "<range>" on page 369.
word	Defines a constraint that overrides the one named by ref when generating search suggestions. For details, see "<word>" on page 369.
collection	Defines a constraint that overrides the one named by ref when generating search suggestions. For details, see "<collection>" on page 369.
word-lexicon	Defines a constraint that overrides the one named by ref when generating search suggestions. For details, see "<word-lexicon>" on page 379.

For example, the following overrides the "tag:" prefix, using the range index for the attribute shortname instead of the one for name when using search:suggest:

```
<constraint name="tag">
  <range collation="http://marklogic.com/collation/"
         type="xs:string" facet="true">
    <element ns="my-namespace" name="my-element"/>
    <attribute ns="" name="name"/>
  </range>
</constraint>
<suggestion-source ref="tag">
  <range collation="http://marklogic.com/collation/"
         type="xs:string" facet="true">
    <element ns="my-namespace" name="my-element"/>
    <attribute ns="" name="shortname"/>
    <suggestion-option>descending</suggestion-option>
  </range>
</suggestion-source>
```

12.27 <term>

Specifies handling of empty searches and controls options for how individual terms (that is, terms not associated with a constraint) will be represented when parsing the search.

To control how empty searches (that is, the empty string passed into search:search) are resolved, specify an empty child element with an apply attribute. The value of the apply attribute specifies the behavior for empty searches: a value of all-results specifies that empty searches return everything in the database, a value of no-results (the default) specifies that an empty search returns nothing. Additionally, you can create your own function to handle empty searches. To specify your own function, create a function that returns a cts:query and specify the local-name of your function in the apply attribute, the namespace of the function library module in the ns attribute, and the path to the module in the at attribute.

You can also specify a `default` child element which determines special handling to all terms, so you can have all terms apply a set of rules such as query weighting, specifying a element or attribute value or word query to be used as a default, or specifying a constraint to be applied by default.

Include a `fragment-scope` element to limit the scope of term queries to document or fragment scope. The local definition overrides a global fragment scope on the enclosing `options` element as long as the term definition does not include a constraint definition or reference, or a custom function specification.

Additionally, you can specify zero or more `term-option` elements to control the behavior of the search terms.

For example:

```
<term>
   <empty apply="no-results" />
   <term-option>diacritic-insensitive</term-option>
   <term-option>unwildcarded</term-option>
</term>
```

The following examples show default term specifications for terms with no constraints:

Term element, parsing to a weighted word query.

```
<term xmlns="http://marklogic.com/appservices/search>
    <default>
        <word>
            <weight>2.0</weight>
        </word>
    </default>
    <empty apply="all-results"/>
</term>
```

Term element, parsing to a weighted element word query on element "foo:bar":

```
<term xmlns="http://marklogic.com/appservices/search">
    <default>
        <word>
            <element ns="foo" name="bar"/>
            <weight>2.0</weight>
        </word>
    </default>
    <empty apply="all-results"/>
</term>
```

Term element, parsing to an element-values query scoped to the properties fragment:

```
<term xmlns="http://marklogic.com/appservices/search">
    <default>
```

```
            <value>
                <element ns="http://example.com" name="copyright-year"/>
                <fragment-scope>properties</fragment-scope>
            </value>
        </default>
        <empty apply="all-results"/>
    </term>
```

Term element, parsing to a field word query via an existing field-word constraint:

```
    <term>
        <default ref="my-field-word-constraint"/>
        <empty apply="all-results"/>
    </term>
```

Term element, referencing custom code (`<default>` configuration will be ignored in a conflict).

```
    <term apply="my-term-query" ns="http://custom/search"
       at="/lib/custom.xqy">
        <default ref="my-field-word-constraint"/>
        <empty apply="all-results"/>
    </term>
```

12.28 <transform-results>

Specifies a function to use to process a search result for the snippet output. The default is that each result is formatted using the built-in default snippetting function.

To use a different snippetting function, you must use either `search:search` or `search:resolve`; the `search:snippet` function always uses the default snippeting function. Specify the local-name of the function to pass in as the value of the apply attribute, the namespace as the value of the ns attribute, and the path to the module as the value of the at attribute. You can pass in parameters to the function by specifying zero or more param child elements (the parameters are passed in in the order specified).

In addition to the default snippetting function, there are three other built-in snippetting functions: `apply="raw"` (return the whole node), `apply="empty-snippet"` (return an empty snippet), and `apply="metadata-snippet"` (return the snippet from the specified element in the properties document).

The following example specifies a custom snippetting function:

```
    <transform-results apply="snippet" ns="my-namespace"
                       at="/my-library.xqy"/>
```

The following example specifies a snippet returning the element named my-element in the properties document:

```
    <transform-results apply="metadata-snippet">
      <preferred-elements>
```

```
    <!-- Specify namespace and localname for elements that exist
         in properties documents -->
    <element ns="" name="my-element"/>
  </preferred-elements>
</transform-results>
```

12.29 <tuples>

Identifies values range indexes or lexicons in which to find value co-occurrences. A `<values-response>` of co-occurrences is included in the search response. You can specify range, collection, geospatial, and uri indexes or lexicons.

A `<tuples/>` element can include an `<aggregate/>` child to specify an aggregate builtin or user-defined function to apply to the values. You can also specify one or more options using `<values-options/>`; see `cts:values` for a list of available options.

The following example specifies co-occurrences between editor and author element values. attribute range index and the URI lexicon.

```
<options xmlns="http://marklogic.com/appservices/search">
  <tuples name="editor-author">
    <range type="xs:string"
        collation="http://marklogic.com/collation/">
       <element name="editor" ns=""/>
    </range>
    <range type="xs:string"
         collation="http://marklogic.com/collation/">
       <element name="author" ns=""/>
    </range>
    <lexicon-option>limit=10</lexicon-option>
  </tuples>
</options>
```

12.30 <values>

Identifies a values range index or lexicon with which to constrain the search. A `<values-response>` of matching index or lexicon values is included in the search response. You can specify range, collection, geospatial, and uri indexes or lexicons.

A `<values/>` element can include an `<aggregate/>` child to specify an aggregate builtin or user-defined function to apply to the values. You can also specify one or more options using `<values-options/>`; see `cts:values` for a list of available options.

The following example specifies values from an element attribute range index and from the URI lexicon. For more examples, see `search:values`.

```
<options xmlns="http://marklogic.com/appservices/search">
    <!-- element attribute range index values -->
    <values name="example">
        <range type="xs:integer">
```

```
            <element ns="" name="game"/>
            <attribute ns="" name="runs"/>
          </range>
          <aggregate apply="sum"/>
        </values>

        <!-- uri lexicon values -->
        <values name="uri">
          <uri/>
          <values-option>limit=10</values-option>
        </values>
      </options>
```

13.0 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, be sure to include the sample XQuery code.

Combined Product Notices

MarkLogic Server 7.0-6 and supporting products.
Last updated: December 11, 2015.

COPYRIGHT

Copyright © 2015 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, US 8,892,599, and US 8,935,267.

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, Version 1.0.1p (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/ v5.6.0. 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 and/or the Stack Exchange Network 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, version 1.4, 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. 1.7.4. 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. 2.5. 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. 1.6. 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.

49. Simple JavaScript Templating. Copyright © 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 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.

50. Clarinet. Original work copyright © Isaacs Z. Schlueter, MIT licensed. Copyright © 2011 Nuno Job <nunojob.com>. 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.

51. Sax. Copyright © Isaac Z. Schlueter ("Author"). v5.6.0. All rights reserved. The BSD License. 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 AUTHOR 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 AUTHOR 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. The file "examples/strict.dtd" is licensed by the W3C and used according to the terms of the W3C SOFTWARE NOTICE AND LICENSE. See LICENSE-W3C.html for details.

52. slimScroll v. 1.3.1. Copyright © Piotr Rochala. 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.

53. jeditable v. 1.7.3. Copyright © Mike Tuupola. 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.

54. jQuery Cookie v. 1.3.1. Copyright © Klaus Hartl. 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.

55. xmlsh v. 1.2.4. Copyright © 2008, David A. Lee. 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. Neither the name of the DEI Services Inc. 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.

56. log4j. 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.

57. httpclient, Version 4.5.1. 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, 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.

58. json-path, Version 0.9.2. Copyright © Kalle Stenflo. 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.

59. json-smart, Version 1.2. Copyright © Uriel Chemouni. 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.

60. saxon HE, Version 9.6.0.7. The open-source Saxon-HE product is made available under the Mozilla Public License (the "License"). Prior to Saxon 9.5 MPL Version 1.0 was used; from 9.5 onwards, MPL Version 2.0 is used. You may not use the software except as permitted by the License. You may obtain a copy of the License at http://www.mozilla.org/MPL/. Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

61. stax-utils. Copyright (c) 2004, Christian Niles, unit12.net. Copyright (c) 2004, Sun Microsystems, Inc. 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. Neither the name of the listed copyright holders 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.

62. woodstox, Version 4.2.0. Copyright © Tatu Saloranta. 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.

63. xerces, Version 2.9.1. 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, 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.

64. functx< version 1.0. Copyright © 1991, 1999 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA. This 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 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.

65. AWS Java SDK v. 1.4.2.1. Copyright © Amazon. 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.

66. Ported and modified implementation of Berghel/Roach. Copyright © Massachusetts Institute of Technology. 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.

67. MIT Kerberos v. 1.10.3. Copyright © 1985-2013 by the Massachusetts Institute of Technology. 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 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. Downloading of this software may constitute an export of cryptographic software from the United States of America that is subject to the United States Export Administration Regulations (EAR), 15 CFR 730-774. Additional laws or regulations may apply. It is the responsibility of the person or entity contemplating export to comply with all applicable export laws and regulations, including obtaining any required license from the U.S. government. The U.S. government prohibits export of encryption source code to certain countries and individuals, including, but not limited to, the countries of Cuba, Iran, North Korea, Sudan, Syria, and residents and nationals of those countries. Documentation components of this software distribution are licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. (http://creativecommons.org/licenses/by-sa/3.0/). Individual source code files are copyright MIT, Cygnus Support, Novell, OpenVision

Technologies, Oracle, Red Hat, Sun Microsystems, FundsXpress, and others. Project Athena, Athena, Athena MUSE, Discuss, Hesiod, Kerberos, Moira, and Zephyr are trademarks of the Massachusetts Institute of Technology (MIT). No commercial use of these trademarks may be made without prior written permission of MIT. "Commercial use" means use of a name in a product or other for-profit manner. It does NOT prevent a commercial firm from referring to the MIT trademarks in order to convey information (although in doing so, recognition of their trademark status should be given).

68. OpenLDAP v. 2.4.31. Redistribution and use of this software and associated documentation ("Software"), with or without modification, are permitted provided that the following conditions are met: 1. Redistributions in source form must retain copyright statements and notices, 2. Redistributions in binary form must reproduce applicable copyright statements and notices, this list of conditions, and the following disclaimer in the documentation and/or other materials provided with the distribution, and 3. Redistributions must contain a verbatim copy of this document. The OpenLDAP Foundation may revise this license from time to time. Each revision is distinguished by a version number. You may use this Software under terms of this license revision or under the terms of any subsequent revision of the license. THIS SOFTWARE IS PROVIDED BY THE OPENLDAP FOUNDATION AND ITS CONTRIBUTORS "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 OPENLDAP FOUNDATION, ITS CONTRIBUTORS, OR THE AUTHOR(S) OR OWNER(S) OF THE SOFTWARE 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. The names of the authors and copyright holders must not be used in advertising or otherwise to promote the sale, use or other dealing in this Software without specific, written prior permission. Title to copyright in this Software shall at all times remain with copyright holders. OpenLDAP is a registered trademark of the OpenLDAP Foundation. Copyright 1999-2003 The OpenLDAP Foundation, Redwood City, California, USA. All Rights Reserved. Permission to copy and distribute verbatim copies of this document is granted.

69. Cyrus Sasl (on Solaris for OpenLDAP) v. 2.1.23. Copyright (c) 1994-2012 Carnegie Mellon 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: 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. The name "Carnegie Mellon University" must not be used to endorse or promote products derived from this software without prior written permission. For permission or any legal details, please contact: Office of Technology Transfer, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213-3890, (412) 268-4387, fax: (412)

268-7395, tech-transfer@andrew.cmu.edu 4. Redistributions of any form whatsoever must retain the following acknowledgment: "This product includes software developed by Computing Services at Carnegie Mellon University (http://www.cmu.edu/computing/)." CARNEGIE MELLON UNIVERSITY DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY BE LIABLE FOR 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.

70. jena- core, Version 2.10.0Copyright © 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.

71. jena-iri, Version 0.9.5 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, 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.

72. jena-arq, Version 2.10.0 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, 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.

73. xerceslmpl, Version 2.10.0 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, 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.

74. commons-modeler, Version 2.0.1. 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, 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.

75. slf4j-api, Version 1.7.5. Copyright (c) 2004-2013 QOS.ch. 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, 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.

76. slf4j-log4j12, Version 1.7.5. Copyright (c) 2004-2013 QOS.ch. 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, 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.

77. xml-apis v. 1.4.01. 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, 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.

78. angular (bower) v. 1.2.20. Copyright © Google. 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 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.

79. angular-bootstrap (bower) v. 0.11.0. Copyright © AngularUI Team. 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 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.

80. angular-cookies (bower) v. 1.2.20. Copyright © Google. 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 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.

81. angular-marked (bower) v. 0.0.12. Copyright © J. Harshbarger. 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 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.

82. angular-mocks (bower) v. 1.2.20. Copyright © Google. 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 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.

83. angular-sanitize (bower) v. 1.2.0. Copyright © Google. 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 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.

84. angular-ui-router (bower) v. 0.2.10. Copyright © AngularUI Team. 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 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.

85. angular-ui-router.stateHelper (bower) v. 1.1.0. Copyright © Mark Lagendijk. 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 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.

86. body-parser (npm) v. 1.5.0. Copyright © Jonathan Ong and Douglas Christopher Wilson. 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 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.

87. bootstrap-sass-official (bower) v. 3.2.0. Copyright © Twitter, Inc. 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 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.

88. chai (bower & npm) v. 1.9.1. Copyright © Jake Luer. 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 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.

89. chai-as-promised (bower & npm) v. 4.1.1. Copyright © Domenic Denicola. Licensed under the WTFPL Version 2 license as available at http://www.wtfpl.net/.

90. chalk (npm) v. 0.5.1. Copyright © Sindre Sorhus. 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 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.

91. connect (npm) v. 3.1.0. Copyright © TJ Holowaychuk. 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 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.

92. connect-livereload (npm) v. 3.1.0. Copyright © Andi Neck. 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 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.

93. connect-modrewrite (npm) v. 0.5.11. Copyright © Tingan Ho. 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 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.

94. cucumber (npm) v. 0.4.1. Copyright © Julien Biezemans. 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 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.

95. del (npm) v. 0.1.1. Copyright © Sindre Sorhus. 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 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.

96. duplexer (npm) v. 0.1.1. Copyright © Raynos. 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 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.

97. es5-shim (bower) v. 4.0.0. Copyright © Kristopher Michael Kowal and contributors. 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 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.

98. event-stream (npm) v. 3.1.7. Copyright © Dominic Tarr. 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 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.

99. express (npm) v. 4.7.2. Copyright © TJ Holowaychuk. 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 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.

100. globule (npm) v. 0.2.0. Copyright © Ben Alman. 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 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.

101. gulp (npm) v. 3.8.6. Copyright © Fractal. 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 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.

102. gulp-browserify (npm) v. 0.5.0. Copyright © Robo. 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 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.

103. gulp-clone (npm) v. 1.0.0. Copyright © Mario Casciaro. 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 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.

104. gulp-debug (npm) v. 0.3.0. Copyright © Sindre Sorhus. 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 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.

105. gulp-embedlr (npm) v. 0.5.2. Copyright © Stian Veum Møllersen. 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 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.

106. gulp-filelog (npm) v. 0.2.0. Copyright © Adam Ayres. 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 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.

107. gulp-filter (npm) v. 0.4.1. Copyright © Sindre Sorhus. 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 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.

108. gulp-if (npm) v. 1.2.4. Copyright © Rob Richardson. 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 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.

109. gulp-ignore (npm) v. 1.2.0. Copyright © Rob Richardson. 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 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.

110. gulp-inject (npm) v. 0.4.1. Copyright © Joakim Bengtson. 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 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.

117. gulp-jscs (npm) v. 0.6.0. Copyright © Sindre Sorhus. 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 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.

111. gulp-jshint (npm) v. 1.7.1. Copyright © Spencer Alger. 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 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.

112. gulp-load-plugins (npm) v. 0.5.3. Copyright © Jack Franklin. 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 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.

113. gulp-match (npm) v. 0.2.0. Copyright © Rob Richardson. 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 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.

114. gulp-mocha-phantomjs (npm) v. 0.3.0. Copyright © Rui Hu . 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 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.

115. gulp-plumber (npm) v. 0.6.4. Copyright © Vsevolod Strukchinsky. 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 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.

116. gulp-rename (npm) v. 1.2.0. Copyright © Hector Guillermo Parra Alvarez. 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 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.

117. gulp-ruby-sass (npm) v. 0.6.6. Copyright © Sindre Sorhus. 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 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.

118. gulp-sass (npm) v. 0.7.2. Copyright © David Manning. 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 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.

119. gulp-template (npm). v. 0.1.1. Copyright © Sindre Sorhus. 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 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.

120. gulp-util (npm) v. 3.0.0. Copyright © Fractal. 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 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.

121. gulp-watch (npm) v. 0.6.8. Copyright © Vsevolod Strukchinsky. 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 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.

122. highcharts-ng (bower) v. 0.0.6. Copyright © Barry Fitzgerald. 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 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.

123. highlight.js (bower) v. 8.0.0. Copyright © Ivan Sagalaev. 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 highlight.js 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 REGENTS 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 REGENTS AND 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.

124. istanbul-middleware (npm) v. 0.2.0. Copyright © Yahoo! Inc. 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 Yahoo! Inc. nor the names of the 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 HOLDERS AND 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.

125. jquery (bower) v. 2.1.1. Copyright © jQuery Foundation and other contributors. 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 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.

126. jshint-stylish (npm) v. 0.4.0. Copyright © Sindre Sorhus. 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 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.

127. jsonschema (npm) v. 1.0.0. Copyright © Tom de Grunt. 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 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.

128. lazypipe (npm) v. 0.2.2. Copyright © Phil DeJarnett. 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 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.

129. lodash (bower & npm) v. 2.4.1. Copyright © John-David Dalton. 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 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.

130. main-bower-files (npm) v. 1.0.1. Copyright © Christopher Knötschke. 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 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.

131. map-stream (npm) v. 0.1.0. Copyright © Dominic Tarr. 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 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.

132. marked (bower) v. 0.3.2. Copyright © Christopher Jeffrey. 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 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.

133. mocha (bower) v. 1.20.1. Copyright © TJ Holowaychuk. 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 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.

134. modernizr (bower) v. 2.8.3. Copyright © Modernizr. 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 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.

135. moment.js (bower) v. 2.7.0. Copyright © Iskren Ivov Chernev. 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 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.

136. ngMarkdown (bower) v. 1.1.0. Copyright © Grumpy Wizards. 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 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.

137. node-html-encoder (npm) v. 0.0.2. Copyright © strictly-software. 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 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.

138. open (npm) v. 0.0.5. Copyright © J Jordan. 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 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.

139. open-sans-fontface. Copyright © Steve Matteson. 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.

140. phantomjs (npm). v. 1.9.7-14. Copyright © Ariya Hidayat. 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 the <organization> 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 HOLDERS 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.

141. protractor (npm) v. 1.0.0. Copyright © Google. 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 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.

142. proxy-middleware (npm) v. 0.5.0. Copyright © Andrew Kelley. 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 the <organization> 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 HOLDERS 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.

143. q (npm) v. 1.0.1. Copyright © Kristopher Michael Kowal. 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 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.

144. q-io (npm) v. 1.11.3. Copyright © Kristopher Michael Kowal. 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 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.

145. requirejs (bower) v. 2.1.14. Copyright © The Dojo Foundation. 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 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.

146. requirejs-plugins (bower) v. 1.0.2. Copyright © Miller Medeiros. 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 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.

147. selenium-webdriver (npm) v. 2.42.1. Copyright © Selenium Project. 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.

148. semver (npm) v. 3.0.1. Copyright © Isaac Z. Schlueter. 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 AUTHOR 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 AUTHOR 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.

149. serve-static (npm) v. 1.4.2. Copyright © Douglas Christopher Wilson. 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 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.

150. shelljs (npm) v. 0.3.0. Copyright © Artur Adib. 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 the Artur Adib nor the names of the 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 ARTUR ADIB 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.

151. sinon (bower) v. 1.9.0. Copyright © Christian Johansen. 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 the Christian Johansen nor the names of his 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.

152. sinon-chai (bower) v. 2.5.0. Copyright © Domenic Denicola. Licensed under the WTFPL Version 2 license as available at http://www.wtfpl.net/.

153. through2 (npm) v. 0.4.2. Copyright © Rod Vagg. 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 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. Except where noted, this license applies to any and all software programs and associated documentation files created by the Original Author, when distributed with the Software.

154. tiny-lr-fork (npm) v. 0.0.5. Copyright © Mickael Daniel. 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 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.

155. vinyl-fs (npm) v. 0.3.4. Copyright © Fractal. 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 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.

156. bluebird v. 1.2.4. Copyright © Petka Antonov. 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 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.

157. concat-stream v. 1.4.6. Copyright © Max Ogden. 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 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.

158. core-util-is v. 1.0.1. Copyright © Isaac Z. Schlueter. 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 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.

159. deepcopy v. 0.3.3. Copyright © Takafumi Sasage. 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 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.

160. dicer v. 0.2.3. Copyright © Brian White. 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 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.

161. node-multipart v. 0.1.5. Copyright © Tim Smart. 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 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.

162. through2 v. 0.5.1. Copyright © Rod Vagg. 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 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. Except where noted, this license applies to any and all software programs and associated documentation files created by the Original Author, when distributed with the Software.

163. winston v. 0.7.3. Copyright © Charlie Robbins. 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 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.

164. www-authenticate v. 0.6.2. Copyright © Randy McLaughlin. 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 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.

165. yakaa v. 1.0.0. Copyright © Joyent, Inc. and other Node contributors. 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 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.

166. gulp-intercept v. 0.1.0. Copyright © Nik Khilnani. 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 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.

167. gulp-jsdoc v. 0.1.4. Copyright © Web It Up (Olivier Gambier). 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 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.

168. gulp-jshint v. 1.5.3. Copyright © Spencer Alger. 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 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.

169. gulp-mocha v. 0.4.1. Copyright © Sindre Sorhus. 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 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.

170. gulp v. 3.6.1. Copyright © Fractal. 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 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.

171. jsdoc v. 3.3.0-alpha5. Copyright © Michael Mathews and the contributors to JSDoc. 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.

172. jshint v. 2.5.1. Copyright © Anton Kovalyov. 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 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.

173. mocha v. 1.18.2. Copyright © TJ Holowaychuk. 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 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.

174. read v. 1.0.5. Copyright © Issac Z. Schlueter. 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 NETBSD FOUNDATION, INC. 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 FOUNDATION 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.

175. should v. 4.0.4. Copyright © TJ Holowaychuk. 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 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.

176. v8 v. 3.24. Copyright © the V8 project authors. 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 Inc. 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.

177. org.codehaus.groovy:groovy-all:2.2.2 v. 2.3.7. Copyright © Pivotal Software. 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.

178. org.springframework.boot:spring-boot-starter-security v. 1.1.8. Copyright © Pivotal Software. 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.

179. org.springframework.boot:spring-boot-starter-web v. 1.1.8. Copyright © Pivotal Software. 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.

180. org.springframework.boot:spring-boot-starter-jetty v. 1.1.8. Copyright © Pivotal Software. 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.

181. org.springframework.boot:spring-boot-starter-test v. 1.1.8. Copyright © Pivotal Software. 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.

182. org.springframework.security:spring-security-web v. 4.0.0. Copyright © Pivotal Software. 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.

183. org.springframework.security:spring-security-test v. 4.0.0. Copyright © Pivotal Software. 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.

184. org.springframework.security:spring-security-ldap v. 4.0.0. Copyright © Pivotal Software. 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.

185. org.apache.directory.server:apacheds-server-jndi:1.5.5 v. 1.5.5. Copyright © 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.

186. com.jayway.jsonpath:json-path:0.9.1 v. 1.1.0. Copyright © Jayway. 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.

187. bluebird v. 2.3.4. Copyright © 2014 Petka Antonov. 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.

188. body v. 4.5.0. Copyright © 2012 Raynos. 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.

189. compression v. 1.2.0. Copyright © 2014 Jonathan Ong. Copyright © 2014 Douglas Christopher Wilson. 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.

190. connect-timeout v. 1.4.0. Copyright © 2014 Jonathan Ong. Copyright © 2014 Douglas Christopher Wilson. 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.

191. cookie-parser v. 1.3.3. Copyright © 2014 TJ Holowaychuk. 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.

192. csurf v. 1.6.2. Copyright © 2014 Jonathan Ong. 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.

193. express v. 4.10.1. Copyright © 2009-2014 TJ Holowaychuk. 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.

194. express-session v. 1.9.1. Copyright © 2010 Sencha Inc. Copyright © 2011 TJ Holowaychuk. Copyright © Douglas Christopher Wilson. 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.

195. lodash v. 2.4.1. Copyright © John-David Dalton. 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.

196. yargs v. 1.3.2. Copyright © 2010 James Halliday. 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.

197. moment.js v. 2.7.0. Copyright © 2011-2014 Tim Wood, Iskren Ivov Chernev. 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.

198. node-monitor v. 0.6.10. Copyright © 2010-2014 Loren West and other contributors. 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.

199. passport v. 0.2.1. Copyright © 2011-2014 Jared Hanson. 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.

200. passport-Idapauth v. 0.2.2. Copyright © 2013 Vese Poikajärvi. 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.

201. chai v. 1.9.1. Copyright © Jake Luer. 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.

202. chai-as-promised. Copyright © Domenic Denicola. Licensed under the WTFPL Version 2 license as available at http://www.wtfpl.net/.

203. chalk v. 0.5.1. Copyright © Sinde Sorhus. 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.

204. duplexer v. 0.1.1. Copyright © 2012 Raynos. 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.

205. event-stream v. 3.1.7. Copyright © 2011 Dominic Tarr. 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.

206. gulp v. 3.8.7. Copyright © 2014 Fractal. 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.

207. gulp-if v. 1.2.4. Copyright © Rob Richardson. 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.

208. gulp-jscs v. 1.1.0. Copyright © Sindre Sorhus. 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.

209. gulp-jshint v. 1.8.4. Copyright © Spencer Alger. 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.

210. gulp-load-plugins v. 0.5.3. Copyright © Jack Franklin. 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.

211. gulp-plumber v. 0.6.4. Copyright © Vsevolod Strukchinsky. 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.

212. gulp-util v. 3.0.1. Copyright © Fractal. 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.

213. gulp-watch v. 0.6.9. Copyright © Vsesvolod Struckchinsky. 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.

214. istanbul-middleware v. 0.2.0. Copyright © Yahoo! Inc. All rights reserved. The BSD License. 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 AUTHOR 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 AUTHOR 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.

215. jshint-stylish v. 0.4.0. Copyright © Sindre Sorhus. 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.

216. lazypipe v. 0.2.2. Copyright © Phil DeJarnett. 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.

217. map-stream v. 0.1.0. Copyright © 2011 Dominic Tarr. 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.

218. mocha v. 1.20.1. Copyright © 2011-2014TJ Holowaychuk. 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.

219. monitor-dashboard v. 0.6.7. Copyright © 2010-2014 Loren West and other contributors. 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.

220. nock v. 0.48.2. Copyright © 2011 Pedro Teixeira. 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.

221. supertest v. 0.41.1. Copyright © 2014 TJ Holowaychuk. 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.

222. through2 v. 0.5.1. Copyright © 2013 Rod Vagg. 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. Distributions of all or part of the Software intended to be used by the recipients as they would use the unmodified Software, containing modifications that substantially alter, remove, or disable functionality of the Software, outside of the documented configuration mechanisms provided by the Software, shall be modified such that the Original Author's bug reporting email addresses and urls are either replaced with the contact information of the parties responsible for the changes, or removed entirely. 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. Except where noted, this license applies to any and all software programs and associated documentation files created by the Original Author, when distributed with the Software.

223. vinyl-buffer v. 0.0.0. Copyright © Fractal. 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.

224. vinyl-fs v. 0.3.6. Copyright © Fractal. 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.

225. qs v. 2.2.3. Copyright © 2014 Nathan LaFreniere. All rights reserved. The BSD License. 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 AUTHOR 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 AUTHOR 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.

226. gradle v. 2.1. Copyright © gradleware. 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.

227. sqlite v. 3.8.7. Copyright © D. Richard Hipp. The author or authors of this code have dedicated any and all copyright interest in this code to the public domain.

228. The MarkLogic Connector for Hadoop contains Cloudera . CDH 5.4.1 and consists of Apache Hadoop and additional key open source projects. Copyright March 18, 2015 You may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.cloudera.com/content/cloudera/en/downloads/cdh/cdh-5-4-1.html.

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.

229. The MarkLogic Connector for Hadoop contains Horton Data Platform HDP 2.2 HDFS. Copyright October 15, 2014, You may not use this file except in compliance with the License. You may obtain a copy of the License at http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.2.4/index.html

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.

230. The MarkLogic MLCP Connector for Hadoop contains Apache Hadoop 2.6.0 Copyright 2014 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: https://hadoop.apache.org/releases.html#18+November%2C+2014%3A+Release+2.6.0+available

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.

231. libgeohash. Copyright © 2010, SimpleGeo. 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 the SimpleGeo 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.

232. Apache Jena (ATQ) v. 2.13.0. Copyright © Isaac Z. Schlueter ("Author"), 2011 Nuno Job. All rights reserved. The BSD License. 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 AUTHOR 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 AUTHOR 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.

234. junit v. 4.12. Copyright © 2002-2015 JUnit. All Rights Reserved. You may obtain a copy of the License at http://junit.org/license.html . EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement , including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

235. logback v. 1.1.3. Copyright © 2004-2013 QOS.ch 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, 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.

236. slf4j v. 1.7.12. Copyright © 1999-2012, QOS.ch. All rights reserved. You may obtain a copy of the License at http://www.eclipse.org/legal/epl-v10.html. EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement , including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

237. Sesame Runtime v. 2.8.5. Copyright © Aduna (http://www.aduna-software.com/) 2001-2013 All rights reserved. You may obtain a copy of the License at https://github.com/ansell/openrdf-sesame/blob/master/core/LICENSE.txt. 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.

238. commons-collections v. 3.2.1. Copyright © Apache. 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.

239. commons-csv v. 1.2. Copyright © Apache. 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.

240. commons-httpclient v. 3.1, 4.3, and 4.5.1. Copyright © Apache. 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.

241. commons-io v. 2.4. Copyright © Apache. 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.

242. Hadoop Package v. 2.6.0. Copyright © Apache. 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.

243. htrace-core v. 3.0.4. Copyright © Apache. 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.

244. jackson-jaxrs, jackson-xc v. 1.9.13. Copyright © Apache. 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.

245. jersey-client, jersey-core v. 1.9. Copyright © Jersey. All Rights Reserved. You may obtain a copy of the License at https://jersey.java.net/license.html. COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

246. junit v. 4.11. Copyright © 2002-2015 JUnit. All Rights Reserved. You may obtain a copy of the License at http://junit.org/license.html . EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement , including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

247. xml-apis v. 1.4.01. Copyright © Apache. 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.

248. Sesame Runtime v. 2.8.7 Copyright © Aduna (http://www.aduna-software.com/) 2001-2013 All rights reserved. You may obtain a copy of the License at https://github.com/ansell/openrdf-sesame/blob/master/core/LICENSE.txt. 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.

249. Apache Commons CSVParser v. 1.2. Copyright © Apache. 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.

250. httpcore v. 4.4.3. Copyright © Apache. 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.

251. resolver v. 1.2. Copyright © Apache. 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.

252. stax2-api v. 3.1.1. Copyright © 2004, Christian Niles, unit12.net. Copyright © 2004, Sun Microsystems, Inc. All rights reserved. You may obtain a copy of the License at https://wiki.kuali.org/display/KULFOUND/StaxUtils+BSD. 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.

253. commons-logging v. 1.1. Copyright © Apache. 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.

254. xom v. 1.2.6. Copyright © 2002, 2004 Elliotte Rusty Harold. You may obtain a copy of the License at http://www.xom.nu/license.xhtml. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

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.